KUNGL TEKNISKA HÖGSKOLAN
Royal Institute of Technology

Performance Evaluation of Multithreaded Architectures with Data Prefetching

Vladimir Vlassov

Department of Teleinformatics

KUNGL TEKNISKA HÖGSKOLAN
Royal Institute of Technology

Performance Evaluation of Multithreaded Architectures with Data Prefetching

Vladimir Vlassov

The research was partially supported by the Swedish National Board for Industrial Development, NUTEK.

The author is supported by a scholarship from the Wenner-Gren Center Foundation for Scientific Research.

TRITA–IT-R 96:05
ISSN 1103–534X
ISRN KTH/IT/R—06/05—SE

February 1996

Department of Teleinformatics

Abstract

A combination of multithreading with data prefetching allows increased efficiency of large-scale multiprocessors. In this report, we evaluate two prefetching techniques in multi-threaded architectures: switch-on-prefetch and run-on-prefetch. The switch-on-prefetch technique switch a thread context on each prefetch operation. The run-on-prefetch technique overlaps prefetching with computation.

This report presents two basic analytical models of multithreading in combination with prefetching, which allow rough performance prediction on the first stages of top-down system design. Both models are based on a few parameters of the multi-threaded architecture and its workload. The first model is the first-order approximation for efficiency of multi-threaded architectures with prefetching, executing a set of threads with fixed timing parameters. The second model is a closed queuing network of the architecture, which is solved for exponentially distributed timing parameters to illustrate its usage for evaluation of multi-threaded architectures with prefetching. The models are validated by comparision with simulation results.

Contents

List of Figures

1: Introduction

One of the main problems in large-scale multiprocessors is considerable memory latency, the time required to access data located in remote memory. Long memory latencies decrease system efficiency. Three basic techniques have been proposed to avoid or tolerate long memory latency: caching, prefetching and multithreading.

Data caching allows keeping copies of remote data in local memory and decreases remote access ratio. A number of caching techniques, such as non-blocking and prefetching caches, have been developed to eliminate enough of the remote memory accesses and to hide long memory latency [3, 4, 5, 6]. In spite of the memory consistency problem in shared-memory multiprocessors which needs system resources to maintain cache coherency, caching continues to be a subject of considerable interest.

Caching already includes data prefetching, even if a cache is not specially constructed to support prefetching explicitly [4]. Prefetching is used to resolve one or more explicit remote references before the data is actually needed by the running process. Like caching, explicit prefetching provides local access to remote data, which have been requested by prefetch operations executed in advance. Mainly, explicit prefetching is used by compilers as a techniques of code optimization [3, 5, 9]. Its efficiency strongly depends on predictability of remote reference sequences.

Multithreading is a general solution to the latency problem. A number of threads is assigned to the same processing node and shares its resources: processing time, memory, etc. When an active thread becomes suspended because of remote memory access (cache miss or explicit remote reference), the processor performs a context switch, and another thread is scheduled for execution. The suspended thread becomes ready and can be reactivated when requested remote data arrives.

Different combinations of the above techniques are under investigation by various research groups [3, 4, 5, 7, 9, 10]. In this paper, we focus on multi-threaded architectures, MTAs, and propose two basic analytical models for multithreading with data prefetching: (i) first-order approximation for efficiency of prefetching MTAs and (ii) a queuing model of prefetching MTAs. These models can be used to obtain rough results in MTA performance evaluation on the first stage of a top-down system design. Both models are based on a few parameters of an MTA and its workload, and allow predicting the efficiency of the MTA executing a set of statistically identical threads.

The remainder of this report is organized as follows: In section 2 an overview of related research in analytical models for multithreading is given. Basic assumptions for the models are presented in Section 3. Section 4 describes studied prefetching techniques. Section 5 introduces the first-order approximation for efficiency of prefetching MTAs. In Section 6, a queuing model of multi-threaded architecture with/without data prefetching is presented. Section 7 deals with validation of both analytical models by comparison with simulation results. Finally, conclusions are given in Section 8.

2: Related Work

A series of three analytical models of multithreading in a cache-based multiprocessor is reported by Saavedra-Barrera et al. in [12] and [13]. MTA efficiency is specified as a ratio of total useful time to the full time including context switching and idle time caused by communication latency:

$$E = Useful/(Useful + Switching + Idle) \tag{1}$$

The top level model of this series is a first-order approximation for MTA efficiency, which is based on a small set of parameters: a number of identical threads (n), communication latency (L), context switch time (C) and run length (R) which is the number of cycles between two consecutive remote references. All timing parameters, L, R and C, are constant. As it was defined in [12], an MTA is saturated when increasing the number of threads does not affect the MTA efficiency, i.e. there is always a thread ready to execute at each context switch. The minimum number of threads, N_s, required to achieve saturation is called saturation point. This model predicts linear dependence of efficiency on the number of threads until $n \geq N_S$; After the saturation point efficiency as a function of n becomes constant:

$$\left.\begin{array}{rl}
\text{Efficiency of single-threaded architecture, STA:} & E_1 = R/(R+L) \\
\text{Efficiency of multi-threaded architecture, MTA:} & E_n = \min(E_L, E_S) \\
\text{Linear region of MTA-efficiency:} & E_L = nR/(R+L+C) \\
\text{Saturation region of MTA-efficiency:} & E_S = R/(R+C) \\
\text{Saturation point:} & N_S = L/(R+C)+1
\end{array}\right\} \tag{2}$$

In the second model presented in [12] and [13] the run length, R, is assumed to be a geometrically distributed random variable. Timed Petri net notation is used to describe a behavior of multithreaded architecture executing a number of statistically identical threads. A reachability set of a Petri net is represented by a Markov chain, which is solved to derive expressions for MTA efficiency and pseudo-saturation point. The third model takes into account a dependence of the cache miss ratio on the number of running threads and the cache size. This dependence affects the run length distribution, since thread context is assumed to be switched on cache misses. The model results in approximations for the miss ratio and the mean run length represented as functions proportional to some power of *n*.

An analytical model of multithreaded processors with caches derived by Agarwal and presented in [1] aims at dependencies $L(n)$ and $R(n) = 1/m(n)$, where $m(n)$ is the miss ratio. Agarwal assumes fixed timing parameters for each thread. The processor-efficiency model is similar in form to (2). Expressions for communication latency $L(n)$, called network model, and miss ratio $m(n)$, called cache model, were derived and used to predict MTA efficiency. As an alternative to the deterministic model Agarwal proposes to model an MTA by a simple queuing system for finite number of customers (threads). Section 6 of this paper contains a queuing model of MTA similar to the model proposed in [1], however our model is stronger and is applied to data prefetching.

While previous studies ignore an interaction between threads located in different nodes of multiprocessor, the model presented by Nemawarkar et al. in [10] uses closed multi-chain queuing networks to evaluate efficiency of a multithreaded multiprocessor with 2-dimensional mesh structure and distributed shared memory. An Approximate Mean Value Analysis [11] (see also [8]) is used to analyse the closed queuing network and evaluate processor-efficiency.

A number of studies on data prefetching in multithreaded architectures has mainly focused on simulation of different hardware- and software controlled prefetching techniques to explore benefits of data prefetching in shared memory cache-based multiprocessors [3, 4, 5, 6, 9].

This paper proposes two basic analytical models of prefetching in combination with multithreading, which allow making rough performance evaluation in top-down system design.

3: Basic Assumptions

Basic assumptions for analytical models of multithreading presented in this paper are inspired from the top-level model of multithreaded architecture proposed in [12]. The multithreading policies mainly differ in events which cause context switching:

- Cache miss, which causes a remote access.
- Explicit remote reference.
- Explicit context switch instructions.

In a multiple-level cache architecture normally a cache miss in the highest level cache causes context switching. Explicit remote references mainly are issued in threads executed on a multithreaded architecture which supports the distributed memory programming model based on massage passing. An explicit context switching is used in a multithreaded architecture with data prefetching.

Both analytical models presented in this paper, (i) first-order approximation for efficiency of prefetching MTAs and (ii) queuing model of multithreading, can be applied to MTAs with different multithreading policies. Nevertheless, for simplicity, while not losing generality, we assume that the MTA is based on a distributed memory model and supports a strategy of context switching on explicit remote references. An inter-node communication is represented by the value of communication latency, *L*. The interconnection network structure is ignored.

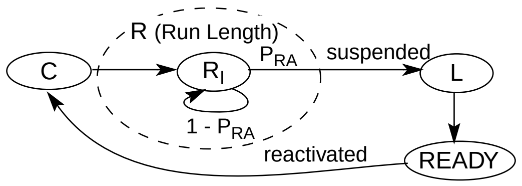


Figure 1: Basic Execution Cycle of a Thread

The workload executed on the architecture is constructed as a set of *n* statistically identical threads. The main parameter of the set of threads is the run interval, R_I, which is the mean number of cycles between two consecutive memory accesses, local or remote (Fig.1). We assume that an instruction with explicit remote reference occurs with the probability P_{RA} and causes a context switch, which takes *C* cycles. The thread is suspended during time *L*, i.e. until data arrives from the remote memory. After that the thread becomes ready and can be reactivated. If the run interval ends by the local access, the thread continues to run. The mean number of cycles between two consecutive remote references (context switches in the MTA), which is called **run length**, *R*, is $R = R_I/P_{RA}$. If we consider $R_I = 1$, then $P_{RA} = 1/R$ is interpreted as a remote reference ratio by analogy to cache miss ratio. The state transition diagram of a thread during run length interval (Fig.1) may include additional states such as cache access and local memory access, according to some programming model (message passing or shared memory). Markov chain analysis can be used to evaluate mean run length in steady-state for different multithreaded architectures (Appendix A).

We assume that a thread may contain prefetch operations to request remote data in advance. A prefetch operation sends a number $m > 0$ of prefetch requests (**prefetch packet**) to the network. We call *m* **length of prefetch packet**. We assume that prefetch requests from one prefetch packet are served in parallel. The probability of prefetching data identified by remote reference is P_P (**prefetching probability**). It specifies the proportion of remote data which can be prefetched. We investigate the efficiency of MTAs with different prefetching techniques, as a function of prefetching probability (P_P) and length of prefetch packets (*m*). We consider the efficiency of an MTA in form of the ratio (1).

4: Data Prefetching in MTA

There are several techniques to exploit prefetching in a multithreading architecture. A good survey of different prefetching schemes can be found in [6]. In this paper we evaluate two prefetching techniques: (i) explicit context switching after prefetch operations (**switch-on-prefetch**); (ii) and overlapping of prefetching with computations (**run-on-prefetch**).

The switch-on-prefetch technique switches on each prefetch operation. The running thread becomes suspended until all requested data arrive. Being resumed the thread executes without suspension on remote references which were prefetched until the next prefetch operation is issued. In caching this technique is called explicit-switch [3].

Overlapping of prefetching with computations in the run-on-prefetch technique allows hiding latency caused by prefetching. An active thread continues to run after a prefetch operation and becomes suspended on a remote reference which is prefetched but the requested data has not yet arrived. In caching this technique is called switch-on-use [3].

4.1: Switch-On-Prefetch Technique

The behavior of the switch-on-prefetch MTA is illustrated by a state transition diagram of a thread depicted on Fig.3.

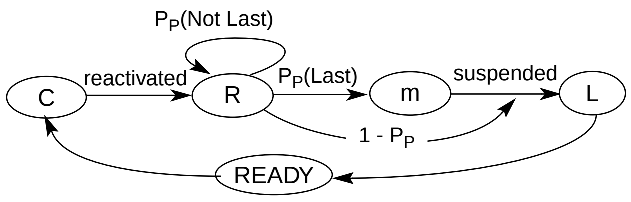


Figure 2: Switch-On-Prefetch: State Diagram for Execution of a Thread

Assume that the execution of a thread starts from prefetch operation marked by *m* where $m > 0$ prefetch requests are sent to the remote destinations. These prefetch requests are served in parallel. The thread is suspended until all requested data arrive (*L*). After reactivation (*C*) the thread is executed (*R*) until a remote reference occurs. If the remote reference was not included in the prefetch packet (probability $1 - P_P$) then the MTA executes a remote fetch operation, causing the running thread to be suspended and another thread scheduled for execution (*C*). If the remote reference was included in the prefetch packet (P_P) then the thread continues to run until it passes $m - 1$ remote references which have been prefetched (*Not Last*). When the last prefetched reference is issued (*Last*) the thread repeats the prefetch operation (*m*).

4.2: Run-On-Prefetch Technique

There are two strategies of overlapping: synchronous and asynchronous. In the case of synchronous overlapping a thread can be suspended when it needs the first prefetched data. The suspended thread becomes ready as soon as all remote data requested by a prefetch packet arrives. Thus the thread waits for all requested data when it needs the first one. With asynchronous overlapping the thread waits for only that prefetched data which is currently needed. Figure 3 shows a state transition diagram of a thread executed with asynchronous overlapping of prefetching with computation.

The MTA sends a prefetch packet (the *m* state) and then executes an active thread during *R* until it needs data from the network. If the remote reference was not included in the prefetch packet (probability $1 - P_P$), the thread initiates a remote access (*L*) and becomes suspended. If the remote reference was included in the prefetch packet (P_P) and the prefetched data has arrived (*Arrived* and *Not Last*) the thread continues to run (*R*). When the last requested data arrives (*Arrived* and *Last*) the thread repeats the prefetch operation (*m*). If requested data has not arrived yet (*Not Arrived*), the executed thread becomes suspended in the blocked state marked by observe latency L_O until either all prefetched data (synchronous overlapping) or only requested data (asynchronous) arrive. In its turn the MTA schedules another thread.

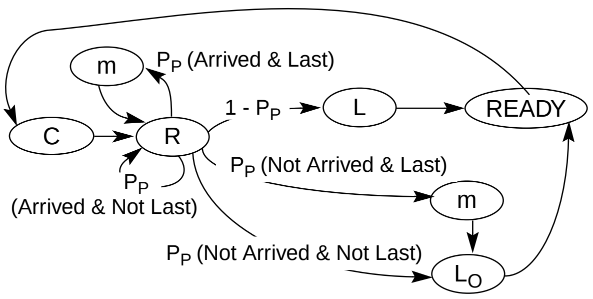

m
P_P (Arrived & Last)
1 - P_P
L
READY
C
R
P_P (Arrived & Not Last)
P_P (Not Arrived & Last)
m
P_P (Not Arrived & Not Last)
L_O

Figure 3: Run-On-Prefetch: Overlapping of Prefetching with Computation

5: First-Order Approximation for Efficiency of MTA

A first-order approximation for the efficiency of MTA without data prefetching as a function of *n* (number of threads) with constant timing parameters *R*, *C* and *L*, was derived in [12] (Equation (2)). This result can be used to predict the efficiency of MTA by using mean values of *R*, *L* and *C*. Figure 4 shows asymptotic bounds E_L and E_S for efficiency of MTA executing statistically identical threads with exponential distributed timing parameters *R*, *L* and *C* (see 6).

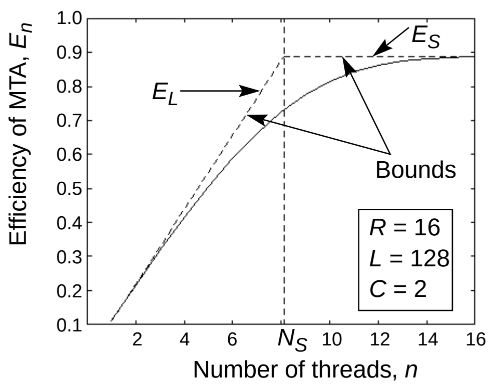


Figure 4: First-Order Approximation for MTA Efficiency

Attempt to define the first-order approximation for efficiency of MTA with data prefetching, considering a thread execution cycle between two prefetch operations.

5.1: Efficiency of Switch-On-Prefetch MTA

Figure 5 illustrates a state transitions diagram of a thread during its execution cycle between two consecutive prefetch operations. The execution cycle starts from prefetch operation marked by *m* after which the thread is suspended (*L*) until requested data arrives. Then it is executed (*R*) until a remote reference occurs. The thread is suspended for a period *L* if this reference is not prefetched $(1 - P_P)$, otherwise the thread continues to run (P_P). When the *m*-th prefetched remote reference is issued the thread repeats the prefetch operation (*m*). The interval T_P is the mean time between two consecutive prefetched remote references:

$$T_P = \frac{R + (L + C)(1 - P_P)}{P_P} \tag{3}$$

The execution cycle includes *m* intervals T_P, latency *L* and a context switch *C*:

$$Full = mT_P + L + C \tag{4}$$

The useful time, the total latency and the switching time of a thread in the execution cycle can be obtained from (4) using (3):

$$\begin{cases} Useful_1 = mR/P_P \\ Latency_1 = (m - mP_P + P_P)L/P_P \\ Switching_1 = (m - mP_P + P_P)C/P_P \end{cases} \tag{5}$$

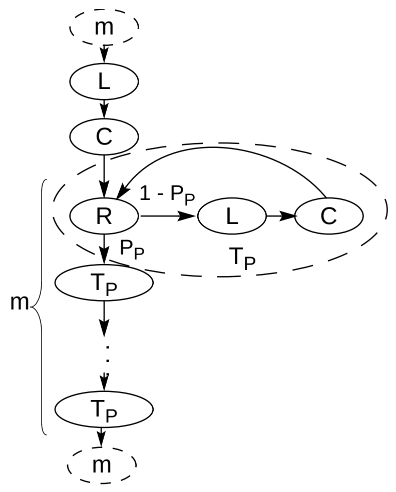


Figure 5: Thread Execution Cycle Between Prefetch Operations in the Switch-On-Prefetch Model

Note that the key assumption of the first-order approximation for efficiency of prefetching MTA is a linear dependence of the efficiency, E_L, on the number of threads, when the MTA is not saturated. This assumption ignores the probabilistic nature of the run-length intervals caused by the probabilistic parameter P_P. As a result of the assumption we can define an optimistic saturation point, n_S, from the inequality $(n-1)(Useful_1 + Switching_1) > Latency_1$, assuming that the total useful and switching time of $n-1$ threads executed when the given thread is suspended, exceeds latency (idle time) of the thread during its execution cycle. However, since the minimum run length of a thread is R, the reliable value for the saturation point, $N_S \geq n_S$, should be defined from the inequality $(n-1)(R+C) > L$, which is valid for non-prefetching MTA (see Equation (2)).

By analogy with (2) we can express the efficiency of the MTA in the following way:

$$\begin{cases} E_1 = \dfrac{Useful_1}{Useful_1 + Latency_1} \\ E_n = \min(E_L, E_S) \\ E_L = \dfrac{n \cdot Useful_1}{Full} \\ E_S = \dfrac{Useful_1}{Useful_1 + Switching_1} \\ n_S = \dfrac{Latency_1}{Useful_1 + Switching_1} + 1 \end{cases} \tag{6}$$

By using (3)-(5) in (6) we obtain the first-order approximation for the efficiency of MTA with explicit context switching after prefetch operations[*]:

$$\begin{cases} E_{1_C} = R_C/(R_C + L) \\ E_{n_C} = \min(E_{L_C}, E_{S_C}) \\ E_{L_C} = nR_C/(R_C + L + C) \\ E_{S_C} = R_C/(R_C + C) \\ n_{S_C} = L/(R_C + C) + 1 \\ N_S = L/(R + C) + 1 \end{cases} \tag{7}$$

where $R_C = \alpha_C R$

$$\alpha_C = m/(m - mP_P + P_P)$$

* Index C in (7) denotes switch-on-prefetch model

We call the coefficient α_C **prefetching factor**. It indirectly characterizes the effect of data prefetching on the efficiency of MTA. The prefetching factor α is an increasing factor for the run length (see 5.5).

5.2: Efficiency of Run-On-Prefetch MTA

Figure 6 illustrates an execution cycle of a thread with synchronous overlapping of prefetching with computations. Since we have assumed that all prefetch requests from prefetch packet are served in parallel, then the asynchronous overlapping behaves as synchronous for the constant communication latency L. The PE sends a prefetch packet (m) and then executes an active thread during R until it needs the first remote data (Fig.6). If that data is prefetched but was has not arrived (P_P), the executed thread is suspended in a state marked by "observe latency" $L - R$ (we assume that $R < L$). After reactivation (C) the thread passes $m - 1$ prefetched remote references without suspension and repeats the prefetch operation (m). If the remote reference after R is not prefetched $(1 - P_P)$, the thread performs a remote fetch operation (L).

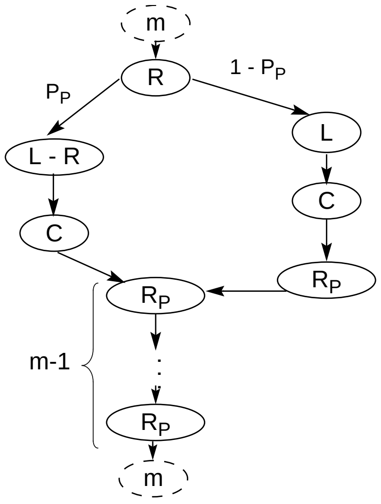


Figure 6: Thread Execution Cycle Between Prefetch Operations in the Run-On-Prefetch Model

By using (3), the full amount of time (useful time, total latency and switching time) of a thread in the execution cycle can be shown to be:

$$\begin{cases} Full = R + (m-1)T_P + (1-P_P)T_P + (1-P_P)L + P_P(L-R) + C \\ Useful_1 = mR/P_P \\ Latency_1 = ((m - mP_P + P_P^2)L - P_P^2 R)/P_P \\ Switching_1 = (m - mP_P + P_P^2)C/P_P \end{cases} \tag{8}$$

By replacing (3) and (8) into (6) we obtain expressions for the efficiency of run-on-prefetch MTA:

$$\begin{cases} E_{1_R} = R_R/(R_R + L_R) \\ E_{n_R} = \min(E_{L_R}, E_{S_R}) \\ E_{L_R} = nR_R/(R_R + C + L_R) \\ E_{S_R} = R_R/(R_R + C) \\ n_{S_R} = L_R/(R_R + C) + 1 \\ N_S = L/(R + C) + 1 \end{cases} \tag{9}$$

where $R_R = \alpha_R R$

$$L_R = L - \beta R$$

$$\alpha_R = m/(m - mP_P + P_P^2)$$

$$\beta = P_P^2/(m - mP_P + P_P^2)$$

Note, that N_S is reliable value for the saturation point, defined for minimum value of thread run length. By analogy with the switch-on-prefetch technique, the coefficient $\alpha_{\boldsymbol{R}}$ is called prefetching factor. By comparing (9) and (7) it is easy to see that the prefetching factor α_R for the run-on-prefetch technique exceeds the factor α_C in switch-on-prefetch.

We call the coefficient β **overlapping factor**. It represents a portion of computation (run length) overlapped with the latency caused by prefetching (see 5.6).

5.3: Limits of MTA Efficiency in Saturation

Note that the linear approximation for efficiency of non-saturated MTA $(n < N_S)$ has increasing error after point $n > L/(mR + C) + 1$ obtained from (2) assuming that run length has the maximum value *mR*. Efficiency functions (7) and (9) can be used only as bounds of real MTA efficiency to make rough preliminary decisions. More detailed investigation of the first-order approximation for efficiency is given in Appendix B. Nevertheless, the first-order approximation allows to predict STA-efficiency and MTA-efficiency in saturation more precisely than in the linear region. The efficiency of prefetching MTA in saturation, when $n \ge N_S$, can be improved by increasing the length of prefetch packets and the proportion of data being prefetched data. Limits of some functions (7) and (9) are:

$$\left.\begin{array}{l}
\lim\limits_{m\to\infty} E_{1_C} = \lim\limits_{m\to\infty} E_{1_R} = \dfrac{R}{R + L(1-P_P)} \\[2ex]
\lim\limits_{m\to\infty} E_{S_C} = \lim\limits_{m\to\infty} E_{S_R} = \dfrac{R}{R + C(1-P_P)} \\[2ex]
\lim\limits_{m\to\infty} R_C = \lim\limits_{m\to\infty} R_R = R/(1-P_P) \\[1ex]
\lim\limits_{m\to\infty} \alpha_C = \lim\limits_{m\to\infty} \alpha_R = 1/(1-P_P)
\end{array}\right\} (P_P < 1) \quad
\left.\begin{array}{l}
\lim\limits_{P_P\to 1} E_{1_C} = mR/(mR + L) \\[1ex]
\lim\limits_{P_P\to 1} E_{1_R} = mR/((m-1)R + L) \\[1ex]
\lim\limits_{P_P\to 1} E_{S_C} = \lim\limits_{P_P\to 1} E_{S_R} = mR/(mR + C) \\[1ex]
\lim\limits_{P_P\to 1} R_C = \lim\limits_{P_P\to 1} R_R = mR \\[1ex]
\lim\limits_{P_P\to 1} \alpha_C = \lim\limits_{P_P\to 1} \alpha_R = m
\end{array}\right\} (m \ne \infty) \quad (10)$$

It is interesting to note that the run-on-prefetch technique has the same limits of efficiency in saturation as switch-on-prefetch. The above expressions can be used to define minimum values of the prefetch parameters (m_γ or $P_{P\gamma}$) required to achieve some percentage, $\gamma \cdot 100\%$, of the efficiency limit. To obtain a solution it is necessary to solve one of the following two equations:

$$E_S(m_\gamma) = \gamma \cdot R/(R + C(1-P_P)),\ \text{when}\ P_P\ \text{is given}$$
$$E_S(P_{P\gamma}) = \gamma \cdot mR/(mR + C),\ \text{when}\ m\ \text{is given}$$

Above equations are by substitution an expression for E_S from (7) reduced to

$$\left.\begin{array}{l}
\alpha(m_\gamma) = \dfrac{\gamma \cdot C}{R(1-\gamma) + C(1-P_P)} \\[2ex]
\alpha(P_{P\gamma}) = \dfrac{\gamma \cdot m \cdot C}{mR(1-\gamma) + C}
\end{array}\right\} \quad (11)$$

For example, assume that prefetching MTA executes a set of threads (n = 9) with parameters L = 128, C = 2 and R = 16 cycles. The portion of remote references which can be prefetched is 65% (P_P = 0.65). According to (2) the saturation point is $N_S = \lceil 8.11 \rceil$, i.e. the MTA is saturated even its workload does not use prefetching. According to (10) the efficiency limit is 0.9581 as $m \to \infty$. For the switch-on-prefetch technique the length of prefetch packets m required to achieve 99% of efficiency limit can be obtain from (11) using expression for α_C presented in (7). The root is $m_{0.99} = \lceil 7.7066 \rceil = 8$. The length of prefetch packets required to achieve the same efficiency in the run-on-prefetch technique is $m_{0.99} = \lceil 5.0093 \rceil = 6$ (obtained from (11) using (9)).

5.4: Efficiency Curves. Comparison of Prefeetching with Non-Prefetching MTA

Figure 7.a,b depicts dependencies of prefetching MTA efficiency on n, m and P_P. Plots $E(m, n)$ and $E(P_P, n)$ show that both the derivative of efficiency in the linear region and the value of efficiency in saturation region increase with increasing prefetching factor. The cause of such trends is increasing the mean value of thread run length due to data prefetching. However the mean run length is a bounded function of prefetch parameters. This is quite obvious, because even if $m \to \infty$ $(P_P < 1)$, execution of a thread is interrupted by remote references which are not prefetched. In the case $P_P \to 1$ $(m \ne \infty)$ all data are prefetched thread contexts are switched on prefetch operations.

Let us compare prefetching MTA, (7) and (9), with non-prefetching MTA (2), assuming that both execute a set of n threads with parameters L = 128 cycles, C = 2 cycles and R = 16 cycles. Expected relative increase, g, in the efficiency of the prefetching MTA against non-prefetching MTA can be calculated as (for run-on-prefetch MTA analogically). Plots $g(m, n)$ and $g(P_P, n)$ in Fig.7.c,d demonstrates the relative efficiency increase as a function of the number of threads, n, for different fixed values of prefetch packet length, m, and prefetching probability, P_P. The function g initially is a constant, until n reaches an optimistic saturation point n_s for the prefetching MTA. After this point, the relative increase graph falls off, because the non-prefetching MTA is not saturated yet, and its efficiency continues to grow in the linear region (Fig.4). After the saturation point of non-prefetching MTA, $N_S = \lceil 8.11 \rceil$, the relative increase graph stabilizes again at values not more than 10%.

Curves of the run-on-prefetch MTA efficiency are similar to switch-on-prefetch. The comparison of the run-on-prefetch MTA efficiency, E_{n_R}, with the efficiency of the switch-on-prefetch MTA, E_{n_C}, is depicted in Fig.8. Plots were obtained according to expression $g(E_{n_R}, E_{n_c}) = ((E_{n_R} - E_{n_c})/E_{n_c}) \cdot 100\%$. For the studied set of parameters ($R = 16$, $L = 128$, $C = 2$ cycles and assuming $P_P \geq 0.5$), the run-on-prefetch technique provides higher efficiency comparing with switch-on-prefetch, when both MTAs are not saturated and the number of threads is less than the optimistic saturation point ($n < 5$). In this case the relative increase in efficiency of the run-on-prefetch MTA is about 10-20% vs. switch-on-prefetch MTA. When both MTAs are saturated ($n > 8$), the relative increase is between 0.25 and 2.5%.

5.5: Mean Run Length in Prefetching MTA

Both prefetching techniques, switch-on-prefetch and run-on-prefetch, allow increasing the mean value of run length, i.e. number of cycles between two consecutive remote references which cause context switching. Assume that a thread is executed forever and specified by run length R (mean number of cycles between two consecutive remote references in a thread), prefetch probability P_P (portion of data which can be prefetched) and m (mean number of prefetch requests generated by each prefetch operation).

First consider the switch-on-prefetch technique. Execution of a thread with context switching after prefetch operation can be represented by a Markov chain with dummy state labelled m (Fig.9.a)*. Being reactivated after suspension (C) the thread is executed (R) until a remote reference is issued. Note that the mean number of remote references between two consecutive prefetch operations is m/P_P, therefore the probability of execution of a prefetch operation is P_P/m. If the reference is not prefetched (probability $(1 - P_P)$) the thread initiates a remote access and becomes suspended (C). If prefetch was initiated the thread either continues to run (probability $P_P(1 - 1/m)$) or repeats the prefetch operation m (probability P_P/m).

By analogy, execution of a thread with synchronous overlapping of prefetching with computations (run-on-prefetch) can be represented by a Markov chain, depicted in Fig.9.b. We analyse only the synchronous overlapping technique assuming that the asynchronous model behaves as the synchronous for constant communication latency. Since the probability of a remote reference between two consecutive prefetch operation is P_P/m, then the probability of prefetched reference is P_P^2/m. Thus, execution of thread (R) is interrupted either by remote references which are not prefetched (probability $(1 - P_P)$) or by a remote reference which is prefetched but the corresponding prefetch request is not yet satisfied (probability P_P^2/m).

The mean run length in prefetching MTAs can be determined by considering both Markov chains as closed chains with a dummy state labelled 0 (Fig.9). The steady-state probability of being at the i-th state of the Markov chain is $P_i = n_i/N$ where n_i is the mean number of repetitions of a state during one loop from R to the dummy state 0, and N is the total number of repetitions of all states in the chain. To define the mean run length the following systems of equations must be solved independently with an assumption $n_0 = 1$:

switch-on-prefetch (Fig.5.a):

$$\begin{cases} n_R = n_0 + n_R P_P(1 - 1/m) \\ n_m = n_R P_P/m \\ n_0 = n_R(1 - P_P) + n_m \\ R_C = n_R R \end{cases}$$

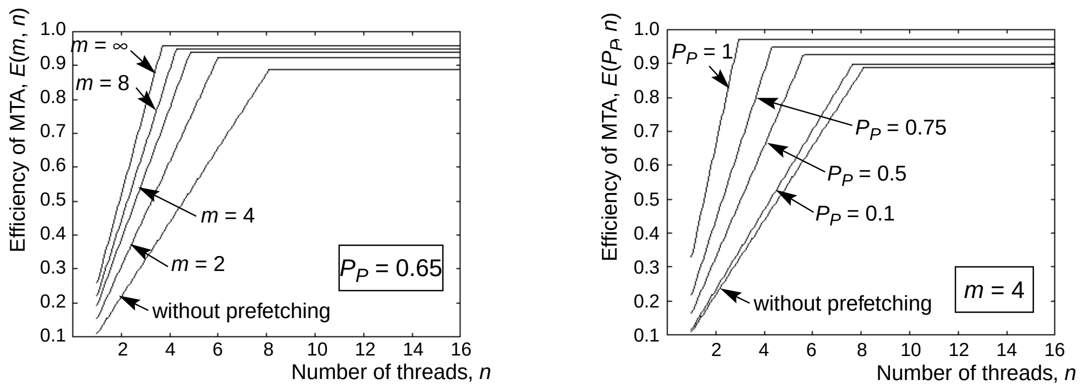
run-on-prefetch (Fig.5.b):

$$\begin{cases} n_R = n_0 + n_R P_P(1 - P_P/m) \\ n_0 = n_R(1 - P_P) + n_R P_P^2/m \\ R_R = n_R R \end{cases}$$

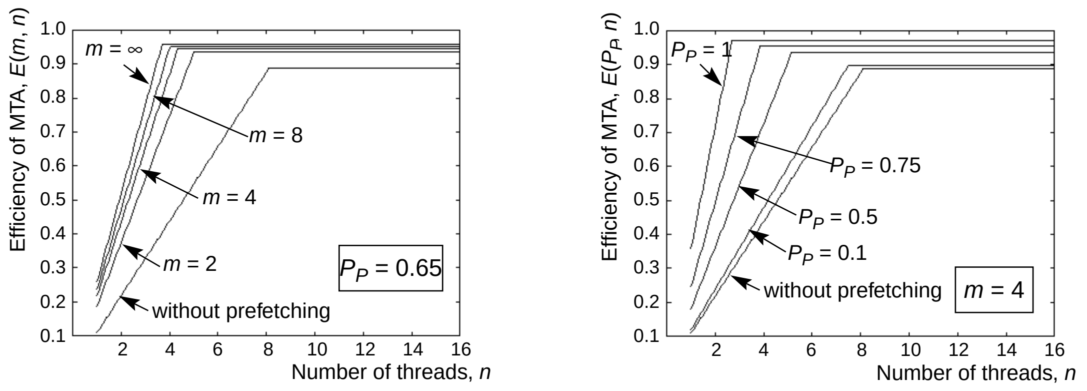
The solutions are

$$\left.\begin{array}{ll} \text{switch-on-prefetch:} & \text{run-on-prefetch:} \\ R_C = \alpha_C R & R_R = \alpha_R R \\ \text{where } \alpha_C = m/(m - P_P m + P_P) & \text{where } \alpha_R = m/(m - P_P m + P_P^2) \end{array}\right\} \quad (12)$$

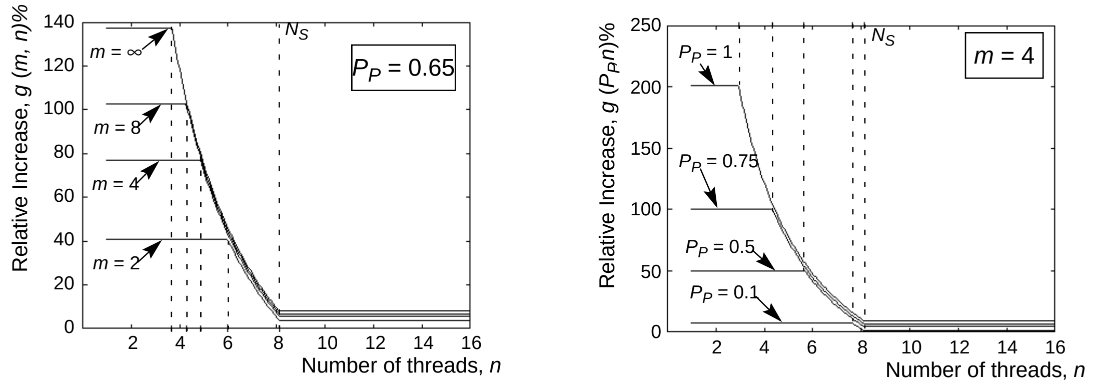
* We assume that the time of prefetch operation is included into R



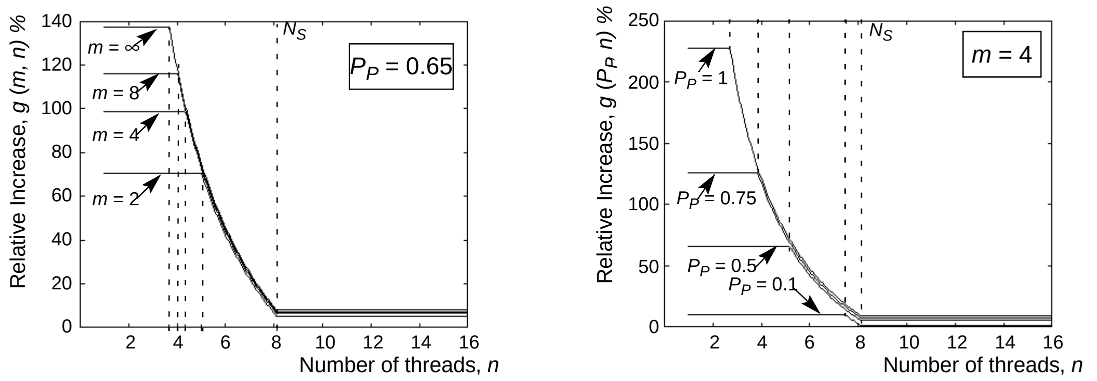
(a) switch-on-prefetch MTA efficiency



(b) run-on-prefetch MTA efficiency



(c) relative increase in efficiency of switch-on-prefetch MTA vs. non-prefetching MTA



(d) relative increase in efficiency of run-on-prefetch MTA vs. non-prefetching MTA

Figure 7: First-Order Approximation for Efficiency of Prefetching MTA (R = 16, L = 128, C = 2)

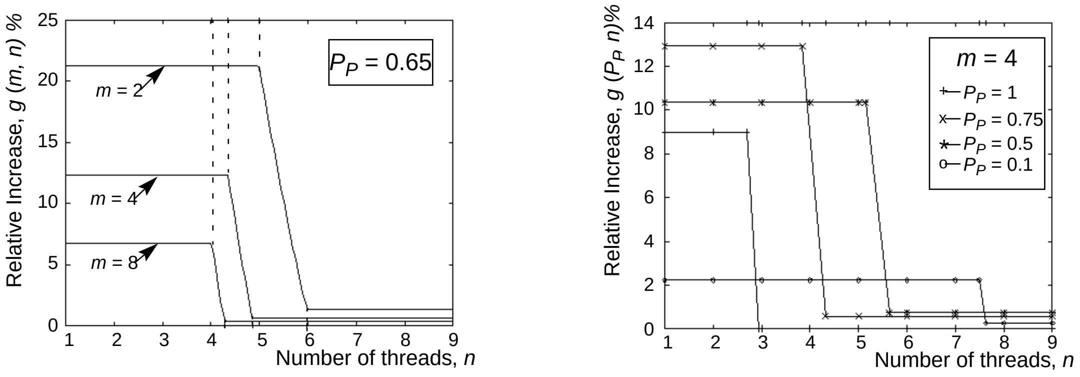


Figure 8: Rrelative Increase in Efficiency of Run-On-Prefetch MTA vs. Switch-On-Prefetch MTA (R = 16, L = 128, C = 2)

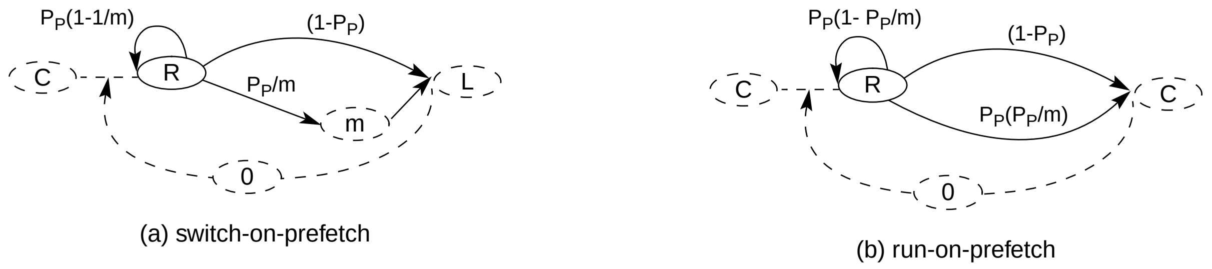
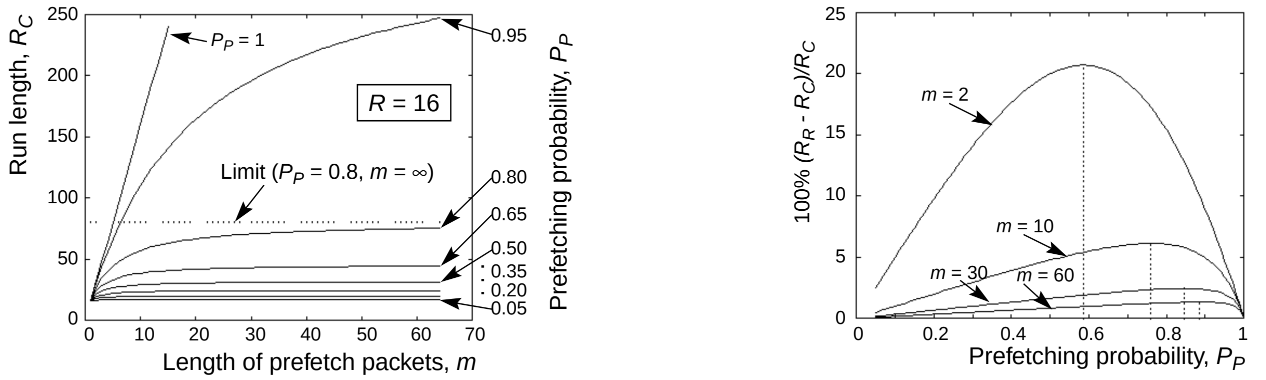


Figure 9: Markov Chains of Run Length in Prefetching MTA

The prefetching factor α is an increasing factor for the run length (Fig.10.a). Comparing α_R and α_C (12) it can be shown that the prefetching factor α_R in the run-on-prefetch technique exceeds the factor α_C in switch-on-prefetch. Figure 10.b depicts the relative increase in the mean of the run length in the run-on-prefetch technique compared to switch-on-prefetch, $g(R_R, R_C) = ((\alpha_R - \alpha_C)/\alpha_C) \cdot 100\%$. It is easy to see that with lower prefetch packet length, m, the relative increase is larger. The function g initially improves, reaches a maximum, and then falls off. It can be shown that the relative increase function, g, achieves its extremum when prefetching probability, P_P, is $(m \pm \sqrt{m})/(m-1) < 1$.



(a) switch-on-prefetch: run length as a function of the prefetch-packet-length and prefetching probability

(b) run-on-prefetch: relative increase in run length vs. switch-on-prefetch MTA

Figure 10: Run Length in Prefetching MTA

5.6: Mean Thread-Idle-Time in Run-On-Prefetch MTA

The run-on-prefetch technique overlaps prefetching with computations and decreases the average value of idle time of a thread (virtual latency). Consider the synchronous overlapping technique. The suspension time interval of a thread can represented by a Markov chain depicted in Fig.11.

The thread becomes idle either for a mean period $L - R_R$ (observed latency) with the probability P_P^2/m (see 5.5) or for a period L (probability $1 - P_P^2/m$). Analysis of the closed Markov chain in Fig.11 gives us an expression for the mean idle time (virtual latency) of a thread for the run-on-prefetch technique:

$$L_R = L - \beta R, \quad \text{where} \quad \beta = P_P^2 / (m - mP_P + P_P^2) \qquad (13)$$

We call the coefficient β the **overlapping factor** which represents a portion of computation (run length) overlapping with latency caused by prefetching.

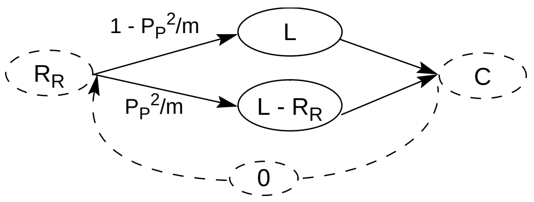


Figure 11: Markov Chain of Thread Idle Time in the Run-On-Prefetch MTA

6: Queuing Model of a Multi-Threaded Architecture

A multi-threaded architecture executing *n* statistically identical threads can be represented as a closed queuing network (Fig.12) with *n* circulating customers (threads). It consists of a queue Q, $1/C$ and $1/R$ servers, and a queuing system with *n* servers $1/L$. The latter assumption means that all remote references from different threads are served in parallel.

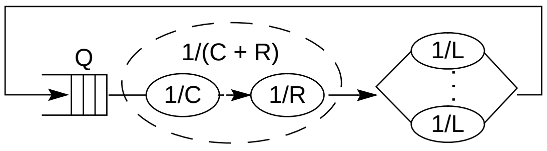


Figure 12: The Closed Queuing Network of MTA

Assume that all timing parameters, *L*, *R* and *C*, have exponential distributions. It is difficult to analyse this network because $1/R$ and $1/C$ servers can not serve more than one customer simultaneously i.e. not more than one thread can be active at any time. The behavior of the network with the above restriction can be described by the continuous-time Markov chain in Fig.13. States of the chain are marked by triple indexes $\{(c, r, l);\ c + r + l = n\}$, which are:

- *c* - number of customers in Q and the $1/C$ server, $c \in \{0, 1, 2, \ldots, n\}$.
- *r* - number of customers in the $1/R$ server, $r \in \{1, 0\}$.
- *l* - the number of customers in $1/L$ servers, $l \in \{0, 1, 2, \ldots, n\}$.

The chain (Fig.13.a) contains $2n + 1$ states and can be analysed computationally using Matlab environment [].

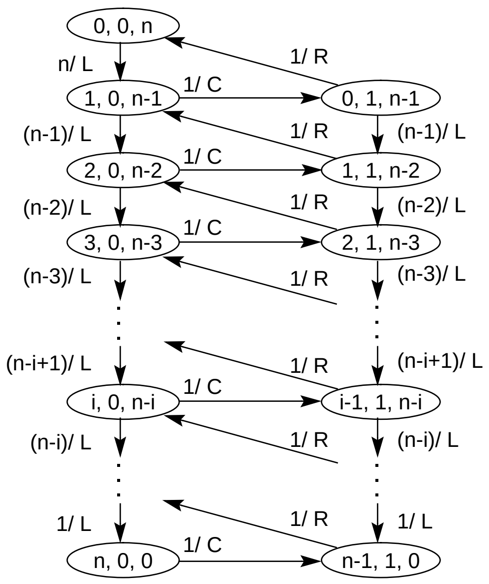


Figure 13: Markov Chain of the MTA Queuing Network

The chain (Fig.13) contains $2n + 1$ states and can be analysed computationally.

To simplify the analytical solution consider $1/C$ and $1/R$ servers (Fig.12), as a closed queuing network consisting of two M/M/1 queuing systems, the first with $1/C$ server and the second with $1/R$ server. Assume that this network serves one circulating thread. The probability of having the thread in the $1/R$ server is the efficiency of the MTA in saturation ($t = \infty$): $E_S = R/(C+R)$.

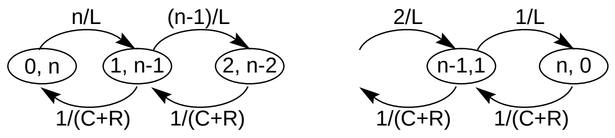


Figure 14: Simplified Markov Chain of the MTA Queuing Network

Next assume that the network (Fig.12) consists of a M/M/1 queuing system with one server (service rate $1/(C+R)$) and a M/M/n queuing system with *n* servers $1/L$. The behavior of this queuing network is represented by the Markov chain (birth-death process) in Fig.14. Each state is marked by a vector (n_1, n_2) where n_i is a number of threads in the *i*-th system. The chain contains $n+1$ states. According the product form analysis the probability of having n_1 threads in the first system and $n_2 = n - n_1$ threads in the second system is:

$$P(n_1, n_2) = \frac{L^{n_2}(C+R)^{n_1}}{n_2!} \cdot \left(\sum_{i=0}^{n} \frac{L^i}{i!}(C+R)^{n-i} \right)^{-1}$$

The probability of having at least one thread in the first system, i.e. one thread in the server $1/(C+R)$, is $(1 - P(0, n))$. Thus the product $(1 - P(0, n))E_S$ is interpreted as the efficiency of the MTA. Queuing theory allows also the derivation of a number of other results for the studied queuing network, such as the mean number of ready threads n_W (mean number of customers in the queue), the mean number of activated and running threads n_{CR} (mean number of customers in the first system), the mean number of suspended threads n_S (mean number of customers in the second system) and the mean waiting time of ready threads t_W (mean waiting time in the queue).

The queuing model of the MTA can be summarized in form of the following expressions:

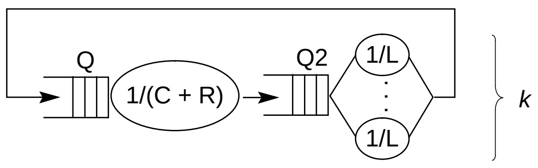
$$\left.\begin{aligned}
\text{Efficiency of the MTA: } & E_n = (1 - P(n))E_S \\
\text{Efficiency limit (saturation): } & E_S = R/(C+R) = \lim_{n \to \infty} E_n \\
\text{Mean number of ready threads: } & n_W = \sum_{i=2}^{n} (i-1)P(n-i) \\
\text{Mean number of active threads: } & n_{CR} = 1 - P(n) \\
\text{Mean number of suspended threads: } & n_S = \sum_{i=1}^{n} iP(i) \\
\text{Mean waiting time of ready threads: } & t_W = \frac{C+R}{1-P(n)} n_W \\
\text{where } & P(x) = \frac{\eta^x}{x!} \cdot \left(\sum_{i=0}^{n} \frac{\eta^i}{i!} \right)^{-1} \\
& \eta = L/(C+R)
\end{aligned}\right\} \qquad (14)$$

An efficiency curve for exponentially distributed timing parameters (14) is bounded by the curve of first-order approximation of the MTA efficiency (2), as it is depicted in Fig.4 (see 5). The curve obtained with the queuing model allows us to conclude that MTA efficiency is almost stabilized when the number of threads increases. It is easy to prove that $(E_n \to E_S)$ as $(n \to \infty)$. The number of threads, n_γ, required to achieve MTA efficiency corresponding to some percentage, $\gamma \cdot (100\%)$, of its limit, can be obtained by repetitive computations of the efficiency E_n for *n* = 1, 2,..., until $E_n \geq \gamma \cdot E_S$.

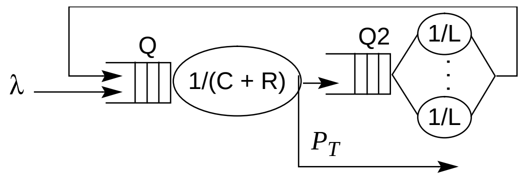
For example, with mean latency L = 128 cycles, a context switch overhead C = 2 cycles, and mean run length R = 16 cycles, the deterministic model gives us the saturation point N_S = 8 threads, where MTA efficiency is predicted to be 0.8889. The queuing model predicts an efficiency value of 0.7243 (82% of its limit) for 8 threads; value of 0.8748 (98%) for 13 threads, and 0.8818 (99%) for 14 threads. These results are close to those predicted in [12].

A pseudo-saturation point, n_S, can be defined with required accuracy ε by repetitive computations of efficiency with n = 1, 2,..., until the difference between current and previous values becomes ≤ ε. The saturation point N_S from the deterministic model (see Eq.(2)) can be used as a starting point for iterations.

We have demonstrated a possible queuing model of a multi-threaded architecture and its analysis. The number of $1/L$ servers (k) can be limited and independent of the number of threads* (Fig.15(a)). An open queuing network (Fig.15(b)) can be used to model an MTA executing a load where the number of threads can be changed by terminating and creating of contexts. For simplicity, streams of newly created threads can be assumed as Poisson arrivals with fixed rate λ. The probability of thread termination, P_T, can be specified as $P_T = R/T$, where T is the mean life-time of a thread.



(a) closed queueing networks of MTA
(number of threads, n, is fixed)



(b) open queueing network of MTA
(number of threads is changeable)

Figure 15: Queuing Networks of MTA

Both closed and open networks (Fig.15) contain $1/L$ servers modelling remote reference service. These servers may be composed in a different way depending on communication network structure. The simplest service disciplines are FIFO, however, one can assume priorities associated to threads. Distribution parameters of service times, L, R and C, may also depend on a thread priority. The efficiency of the MTA in steady state is the probability of having at least one thread in the $1/(C+R)$ server multiplied by the ratio $R/(C+R)$. This multiplication should be done to exclude switching time from useful time. Experts in queuing theory have developed a number of methods (see for example [2, 8, 11]) to analyse such queuing networks, which are similar to the central server model of a time-sharing system [2, 8].

6.1: Usage of the Queuing Model for Evaluation of Efficiency of Prefetching MTA

The expressions (14) obtained with the queuing model can be used for evaluation of efficiency of MTAs with data prefetching. For the switch-on-prefetch MTA the parameter R (run length) in expressions (14) can be simply replaced by the mean run length R_C from equation (12) (see 5.5). Figure 16.a depicts dependencies of the switch-on-prefetch MTA efficiency on workload parameters, such as number of threads, n, length of prefetch packets, m, and proportion of prefetched data, P_P.

For the run-on-prefetch MTA the parameter R (run length) in expressions (14) must be replaced by the mean run length R_R from equation (12) (see 5.5). The parameter L must be replaced by the mean latency L_R from (13) (see 5.6). Figure 16.b depicts efficiency of the run-on-prefetch MTA as functions of workload parameters n, m, and P_P.

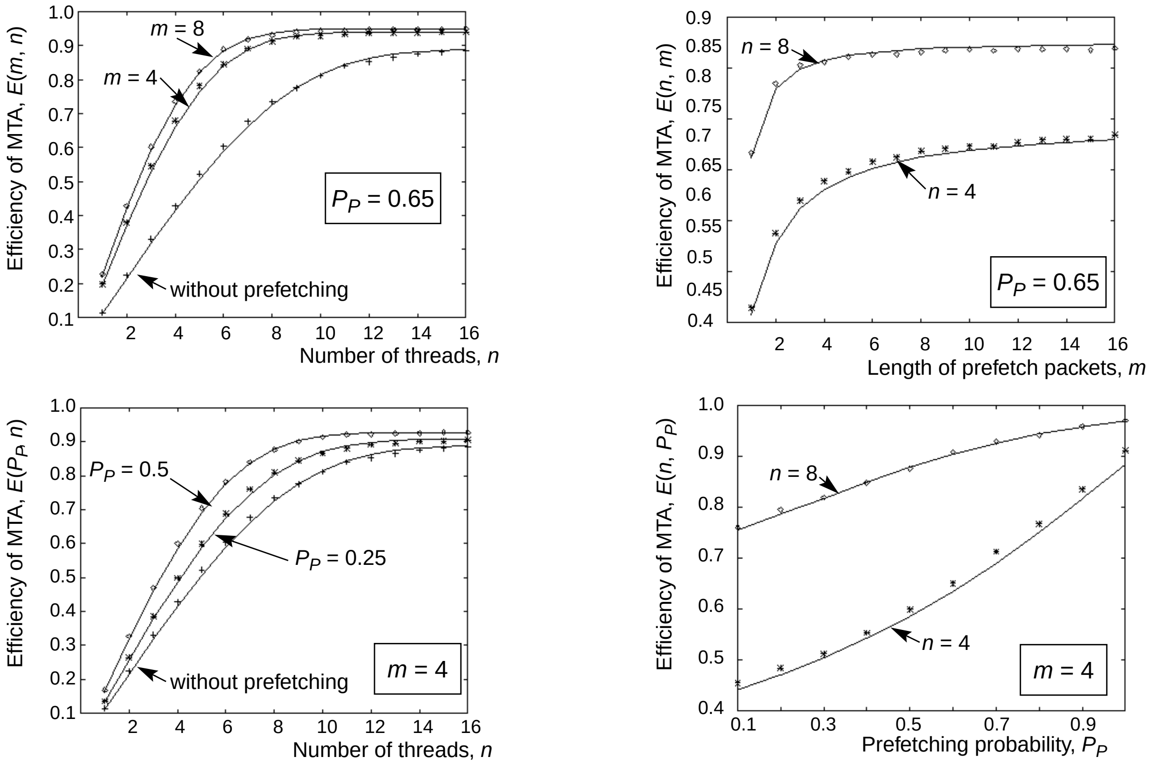
A pseudo-saturation point, n_S, where efficiency is closed to saturation with required error ε, can be found by repetitive computations of efficiency with n = 1, 2,..., until the difference between current and previous values becomes ≤ ε. The saturation point $N_S = L/(C+R)+1$ from the deterministic model can be used as a starting point for iterations.

Plots $E(n, m)$ and $E(n, P_P)$ in Fig.16.a and Fig.16.b show that efficiency of prefetching MTA executing a fixed number of threads can be improved by increasing the prefetching factor (length of prefetch packets or/and proportion of prefetched data) for its workload. However, the efficiency is almost stabilized when the length of prefetch packet, m, reaches some value called here **pseudo-stability point**, m_S. This point can be defined with required accuracy ε by repetitive computations of efficiency with m = 1, 2,..., until the difference between current and previous values becomes ≤ ε.

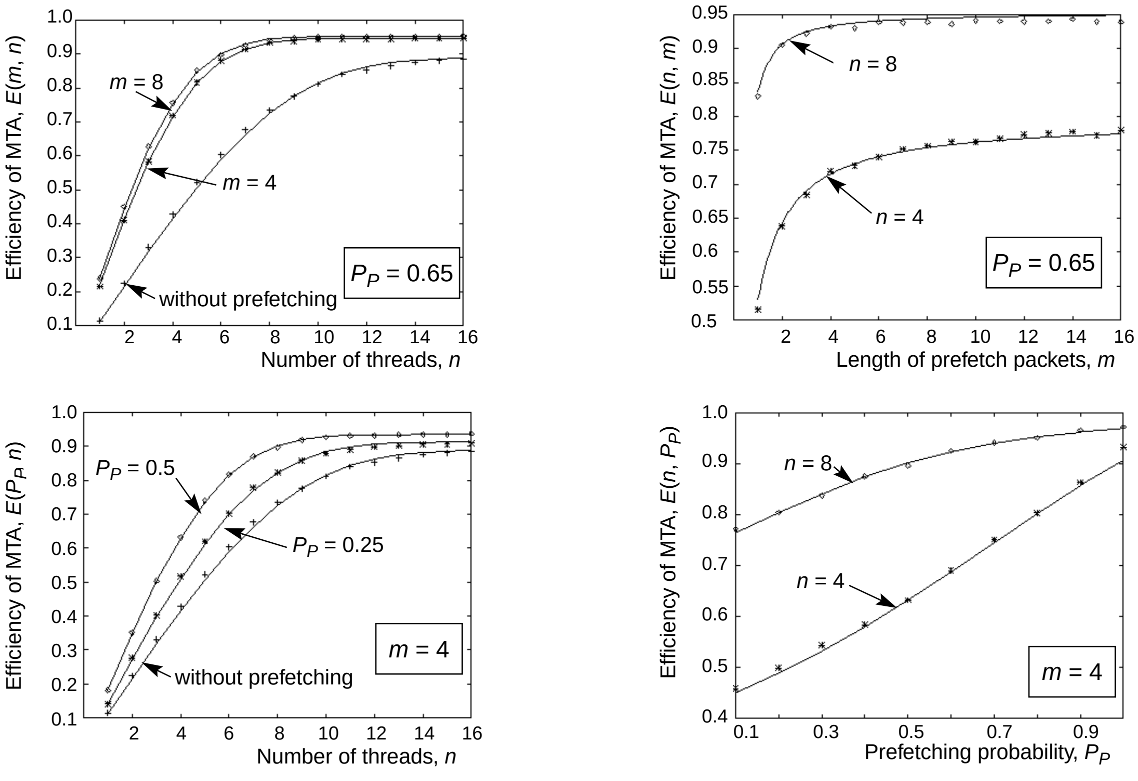
A limit of the function E_n as $m \to \infty$ is (using equations (10) and (14)):

$$\lim_{m \to \infty} E_n = \left(1 - \frac{\eta^n}{n!} \cdot \left(\sum_{i=0}^{n} \frac{\eta^i}{i!}\right)^{-1}\right) E_S$$
$$\text{where} \left(\eta = \frac{L(1-P_P)}{R + C(1-P_P)}\right) \qquad (15)$$
$$\left(E_S = \frac{R}{R + C(1-P_P)}\right)$$

* In the presented analysis we actually assumed that the number of $1/L$ servers is equal to the number of threads



(a) switch-on-prefetch model



(b) run-on-prefetch model

Figure 16: Queuing Model: Efficiency of Prefetching MTA (*R* = 16, *L* = 128, *C* = 2). Symbols correspond to simulation and curves without symbols correspond to the model

A prefetch packet length, m_γ, when MTA efficiency, $E(m_\gamma)$ corresponds to some percentage, $\gamma \cdot (100\%)$, of its limit, can be obtained by repetitive computations of $E(m)$ for $m = 1, 2, \ldots$, until it becomes not less than desired value.

For example, assume that a switch-on-prefetch MTA with mean latency $L = 128$ cycles, and a context switch overhead $C = 2$ cycles, executes 8 threads with data prefetching. Each thread has mean run length $R = 16$ between consecutive remote references, 65% of which are prefetched ($P_P = 0.65$). For the studied parameters, according to (15) the efficiency limit is

0.9537 as $m \to \infty$. Calculating the mean run length, R_C (12), with different values of m and using it in (14) instead of R, we can see that from $m = 8$ (pseudo-stability point) the efficiency of switch-on-prefetch MTA is almost stabilised at the value 0.94 (98.6% of the limit) with an error less than 0.002. For the same set of parameters, the efficiency of run-on-prefetch MTA has the same limit 0.9537 when $m \to \infty$. However, a pseudo-stability point for run-on-prefetch MTA equals 6 requests per prefetch packet.

6.1.1: Prefetching MTA vs. Non-Prefetching MTA

Figure 17.a depicts the relative increase in efficiency of prefetching MTA compared with non-prefetching MTA. Curves of the run-on-prefetch MTA efficiency are similar to switch-on-prefetch. As in the first-order approximation for MTA efficiency, presented in §5, plots $g(m, n)$ and $g(P_P, n)$ show that the advantage of prefetching MTA over non-prefetching MTA degrades when the number of threads, n, increases (compare Fig.7.b with Fig.17.a). This is because both prefetching and non-prefetching MTAs reach their pseudo-saturation points, and their efficiencies are almost stabilized.

Plot $g(n, m)$ in Fig.17.a illustrate the dependence of the relative increase function on length of prefetch packets, m, for different fixed numbers of threads, n. These plots show that he relative increase in efficiency of prefetching MTA vs. non-prefetching MTA initially grows, if the length of prefetch packets, m, increases, but when prefetching MTA reaches its pseudo-stability point, the relative increase is almost stabilized.

The advantage of prefetching MTA against non-prefetching MTA is more sensitive to the value of prefetching probability, P_P (see plots $g(n, P_P)$ in Fig.17.a). The relative increase in efficiency grows when the proportion of prefetched data increases. However, when the number of executed threads increases this dependence becomes weaker.

6.1.2: Run-On-Prefetch MTA vs. Switch-On-Prefetch MTA

By analogy, we compare the efficiency of MTAs with different prefetching techniques. Figure 17.b depicts the relative increase in the efficiency of the run-on-prefetch MTA compared with switch-on-prefetch MTA.

For the studied set of parameters ($R = 16$, $L = 128$, $C = 2$ cycles and assuming $P_P \geq 0.5$), plots $g(m, n)$ and $g(P_P, n)$ demonstrate that the run-on-prefetch technique provides higher efficiency than switch-on-prefetch, when both MTAs are loaded by a number of threads less than the saturation point, as predicted by first-order approximation for efficiency (9 threads in this case). After this point the relative increase in efficiency of the run-on-prefetch MTA is 0.25 and 2.5%.

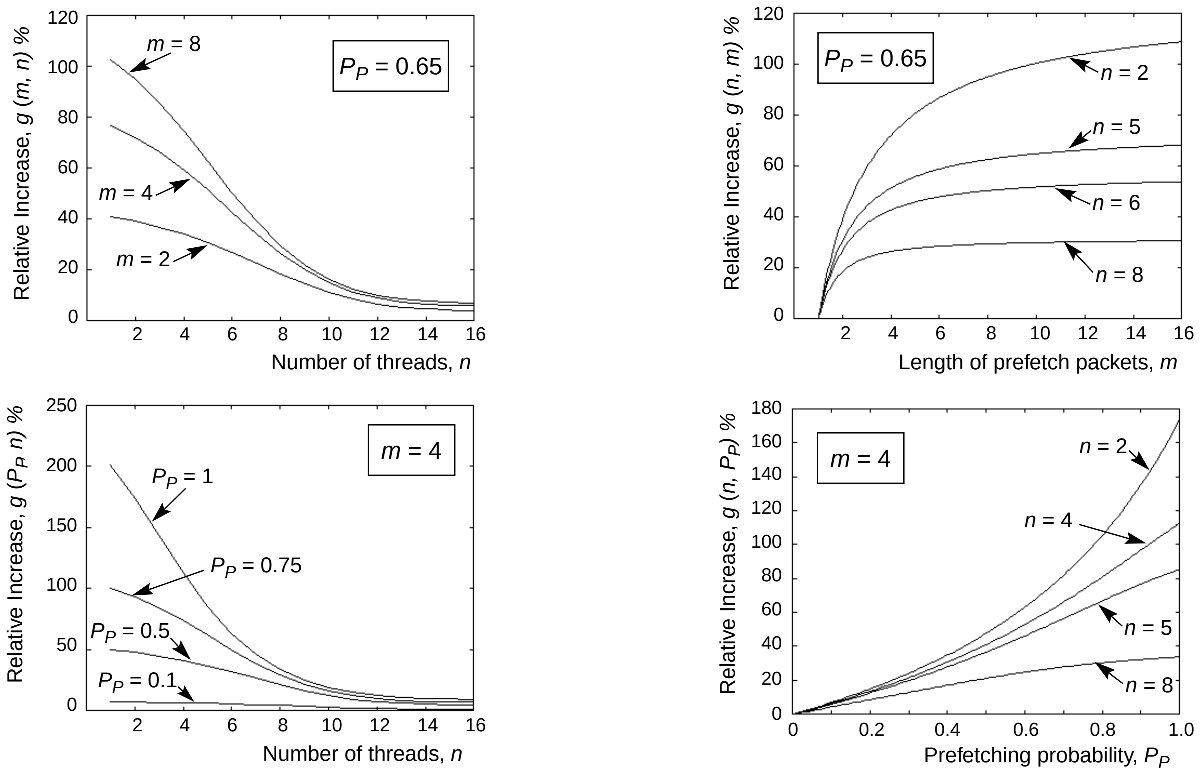
We can conclude also that the advantage of the run-on-prefetch technique over switch-on-prefetch degrades when the length of prefetch packets, m, increases (see plots $g(n, m)$ in Fig.17.b). After pseudo-stability points the relative increase is almost stabilized at 1-6%. On the other hand, the advantage of the run-on-prefetch technique initially grows when the proportion of prefetched data, P_P, increases (see plots $g(n, P_P)$ in Fig.17.b).

7: Validation of the Models

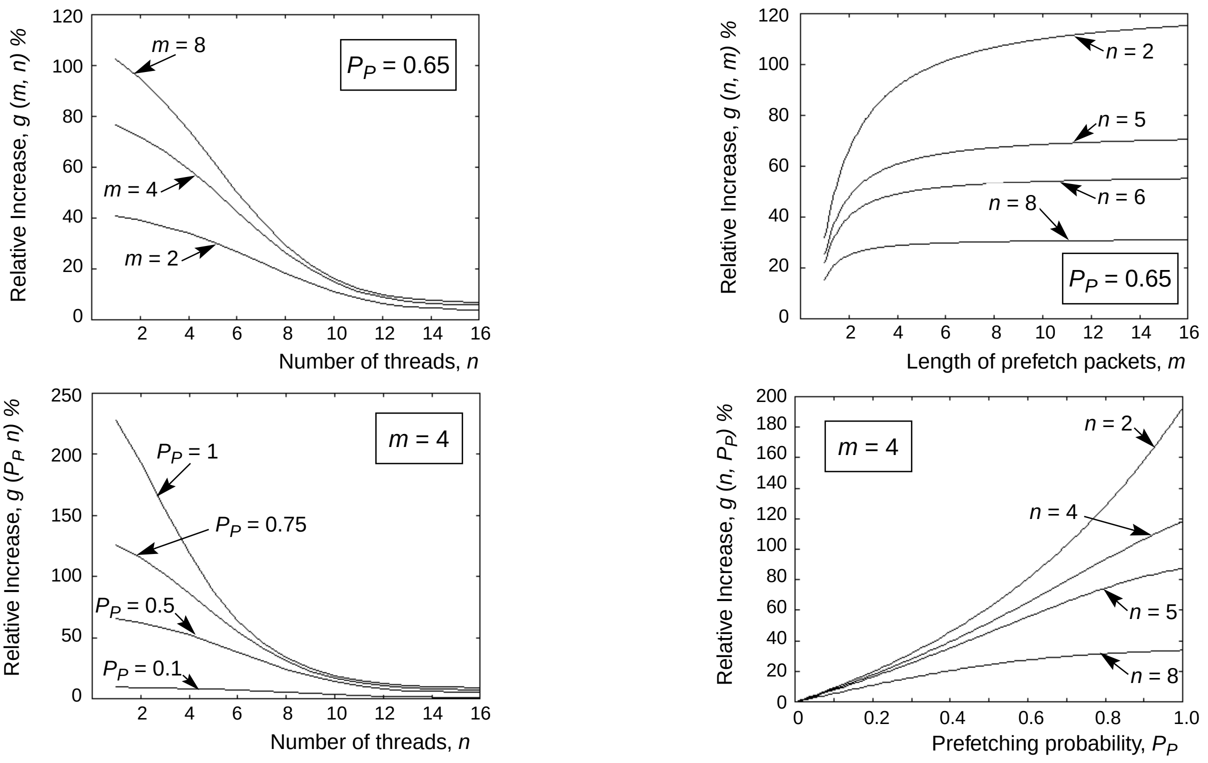
The first-order approximation for efficiency of prefetching MTA is represented by the equations (7) and (9) in 5. The simple solution for the queuing model of multithreading with (without) prefetching is represented by the basic expressions (14) in 6. These expressions can be used to evaluate prefetching MTA using (12) and (13), as described in 6.1. Both models were validated using an MTA simulator presented in [14]. The simulator is based on a Finite State Machine model and generates execution traces of synthetic threads in form of sequences of thread timed states such as running, switching, suspended. While executing, the MTA simulator collects relevant statistics which are used by a plotter to display results of experiments. Model validation compares predicted MTA efficiency, mean run length and mean idle time of threads with results obtained from simulation.

For the first model we report results of validation with the following fixed timing parameters: $L = 128$, $R = 16$ and $C = 2$. Figure 19 shows both analytical and experimental curves of efficiency for prefetching MTAs. It is easy to see that theoretical lines on linear and saturation regions bound the experimental curves. Simulation proves that prefetching MTAs are saturated by the same number of threads as those without prefetching. In all presented results the saturation point equals 9 threads, which corresponds to the saturation point of non-prefetching MTA executing threads with the same parameters.

The queuing model was validated by simulation with the following timing parameters: $L = 128$ (exponential), $R = 16$ (exponential) and $C = 2$. In the presented experiments we do not evaluate efficiency of run-on-prefetch MTA based on asynchronous overlapping of prefetching with computation, because the expressions for mean run length (12) and mean idle time of a thread (13) are valid for asynchronous overlapping only if R and L are fixed. Figure 16 depicts results of the validation which show a good coincidence of analytical and experimental curves.

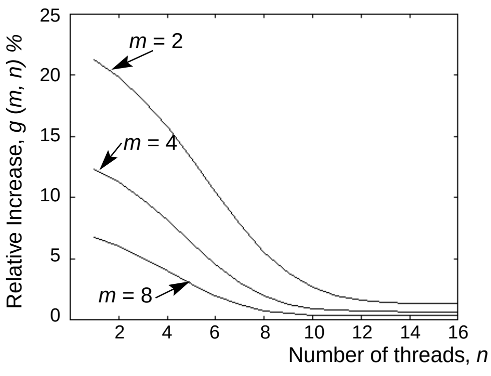


(a) switch-on-prefetch MTA vs. non-prefetching MTA

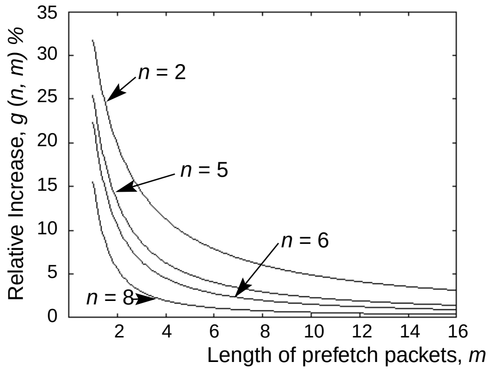


(b) run-on-prefetch MTA vs. non-prefetching MTA

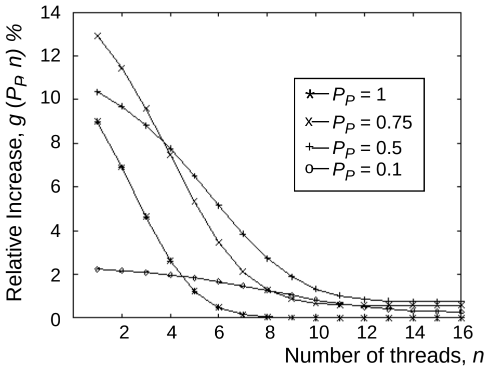
Figure 17: Queuing Model: Relative Increase in Efficiency of Prefetching MTA vs. Non-Prefetching MTA (*R* = 16, *L* = 128, *C* = 2)



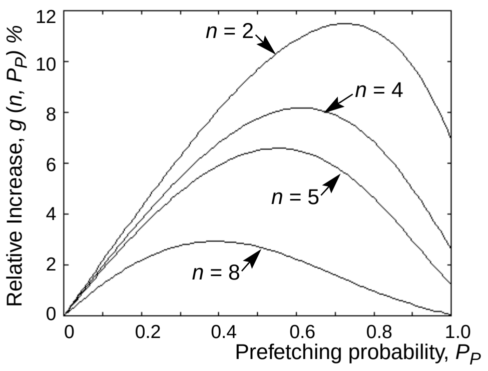
(a) as a function of m and n ($P_P = 0.65$)



(b) as a function of n and m ($P_P = 0.65$)

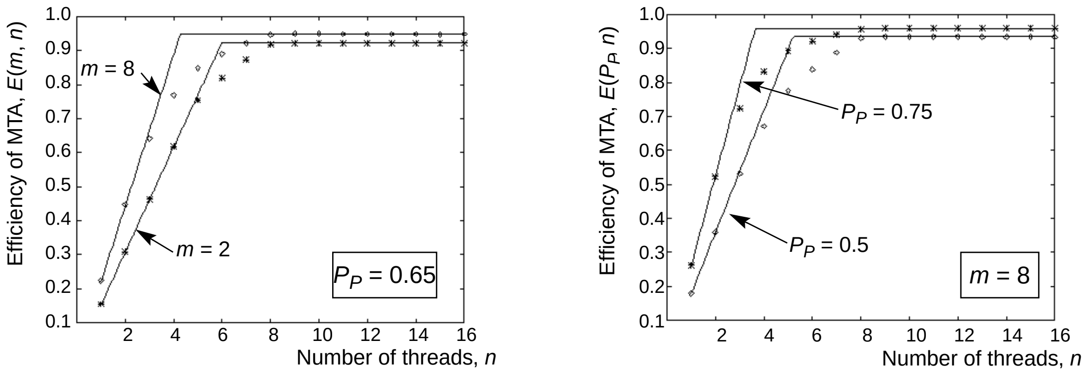


(c) as a function of P_P and n ($m = 4$)

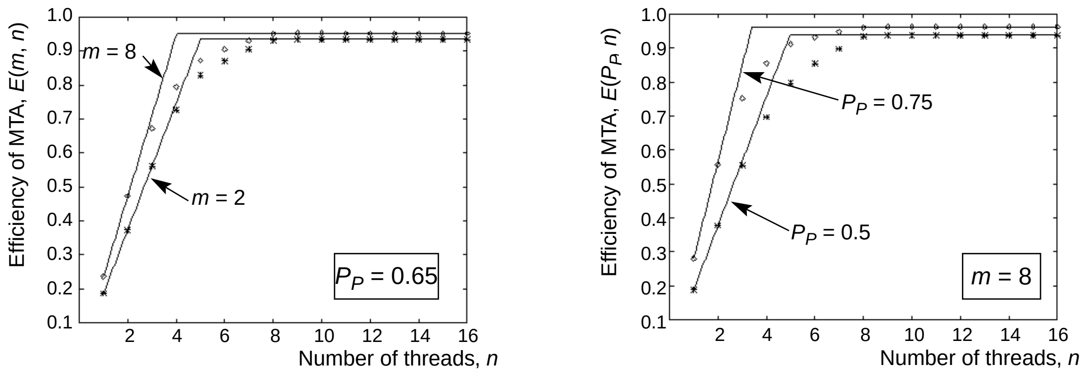


(d) as a function of P_P and n ($m = 4$)

Figure 18: Queuing Model: Relative Increase in Efficiency of Run-On-Prefetch MTA vs. Switch-On-Prefetch MTA (R = 16, L = 128, C = 2)



(a) switch-on-prefetch MTA efficiency



(b) run-on-prefetch MTA efficiency

Figure 19: Validation of the First-Order Approximation for MTA Efficiency (R = 16, L = 128, C = 2) Symbols correspond to simulation and curves without symbols correspond to the model

8: Conclusions

We presented two analytical models of multithreaded architectures with data prefetching. We evaluated two prefetching techniques: switch-on-prefetch, and run-on-prefetch (mainly synchronous overlapping of prefetching with computation). Both presented models can be used to predict the efficiency of a prefetching MTA assuming execution of a fixed number of statistically identical threads. The architecture and its workload are specified in terms of communication latency (L), context switching overhead (C), number of threads (n), number of cycles between two consecutive remote references (R), proportion of data which can be prefetched (P_P), and number of prefetch requests per prefetch operation (m).

The first model is the first-order approximation for MTA efficiency which is based on assumptions that all timing parameters (L, C, R) are constant, the length of prefetch packets (m) is fixed, and prefetched remote references are uniformly distributed in a sequence of remote references within a thread. This model predicts the efficiency of MTA in saturation and efficiency of STA more precisely than the efficiency of non-saturated MTA. It allows predicting a mean run length of a thread with data prefetching and mean idle time of a thread in a run-on-prefetch MTA. The model predicts an optimistic saturation point based on the assumption of linear dependence of efficiency on number of threads in a non-saturated MTA. However, we have demonstrated by simulation that for both prefetching techniques, switch-on-prefetch and run-on-prefetch, a more reliable saturation point is the same as for a non-prefetching MTA executing the same load.

The second model is a queuing network of an MTA with (without) prefetching. To demonstrate the usage of this model, the closed queuing network of the MTA was solved for the case of exponentially distributed timing parameters and a fixed number of statistically identical threads. An open queuing network of the MTA can be used for the case when the number of threads is dynamically changeable.

We proved that the mean run length of a thread in prefetching MTAs grows when one or both prefetching parameters (P_P and m) increase. We have derived expressions for how much the run-length increases (called here prefetching factor) for both prefetching techniques. The run-on-prefetch technique allows overlapping prefetching with computation, and results in decreasing the mean idle time of a thread. The overlapping is characterized by an overlapping factor. The efficiency of MTA can be improved by increasing prefetching and overlapping factors of its workload. However MTA-efficiency is a bounded function of both parameters and almost stabilized when the length of prefetch packets (m) increases over a finite point (called here pseudo-stability point). We have derived equations (first/second order) to define values of prefetch parameters required to achieve a desired efficiency close to the limit.

Acknowledgments

I would like to thank my supervisor professor Lars-Erik Thorelli for his encouragement and support of this work. I am extremely grateful for his reading and comments on this report. Special thanks to my colleague, friend and initiator of this research Dr. Hallo Ahmed for his support and helpful discussions. Many thanks to my colleague Dr. Alexander V. Kraynikov (St.Petersburg's Electrotectical University, Russia) for helping validate the queuing model. Thanks also to my colleague and friends Yuri Ismailov (Computer Communication Systems Lab) and Tarmo Uustalu (Sofware Engineering Lab) for their critical suggestions and comments on the model derivation. Thanks to Abdel-Hlim Smai for his reading and comments on this report.

The research reported here was partially supported by the Swidish National Board for Industrial Development, NUTEK. The author is supported by a scholarship from the Wenner-Gren Center Foundation for Scientific Research.

Appendix A:Markov Models of Run Length

This Appendix presents a technique which can be used for evaluation of the run length (R) for different context switching strategies, depending on the programming model (massage passing or shared memory). We assume that the PE executes a thread during $t = \infty$. To evaluate the mean R in steady-state we introduce a set of Markov chains each of which represents state transitions of a thread during a run length on a particular architecture. The behavior of a Markov chain depends on architectural parameters, such as context switching and data prefetching strategies, cache latency (L_C), local memory latency (L_M). Note that Markov chains presented in this report are basic patterns and can be changed for other architectures.

A.1: The Simplest Markov Chain of Run Length

The simplest Markov chain of R is depicted in Fig.A.1. A thread is specified by a mean value of run interval, R_I, which is the number of cycles between two consecutive memory accesses, local or remote. Remote access occurs with the probability $P_{CS} \neq 0$ (called context switch probability) and can be caused by an explicit remote reference or a local memory miss. In its turn remote access causes a context switch (C).

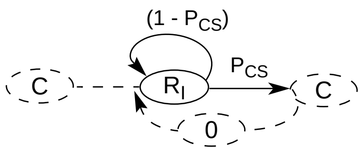


Figure A.1: The simplest Markov Chain of $R_{(1)}$

Assume that the Markov chain is closed with dummy state marked 0 (Fig.A.1). The steady-state probability of being at the i-th state of the Markov chain is $P_i = n_i / N$ where n_i is the mean number of repetitions of a state during one loop from state R to state 0, and N is the total number of repetitions of all states in the chain. The mean run length R, can be derived from the following system of equations with assumption $n_0 = 1$:

$$\begin{cases} n_0 = n_{R_I} P_{CS} = 1 \\ n_{R_I} = n_0 + n_{R_I}(1 - P_{CS}) \\ R_{(1)} = n_{R_I} R_I \end{cases}$$

Thus the mean run length as a function of context switch probability is:

$$R_{(1)} = R_I / P_{CS} \tag{A1}$$

A.2: Run Length in a Message Passing Architecture without Data Prefetching

Consider a strategy of context switching on explicit remote references without data prefetching (message passing architecture). A thread is specified by the mean value of run interval, $R_{I,}$ which is the number of cycles between two consecutive memory accesses, local or remote. The probability of explicit remote reference is P_R (context switch probability). The time required to generate a remote access request is r. The time needed for a load operation is l. The probability of a cache access P_C, the probability of cache miss is P_m. We assume that the thread reactivated after suspension passes through a number of timed states (timing intervals) until a context switch (C) is initiated as a result of an explicit remote reference. Thread state transitions are represented by the Markov chain depicted in Fig.A.2.

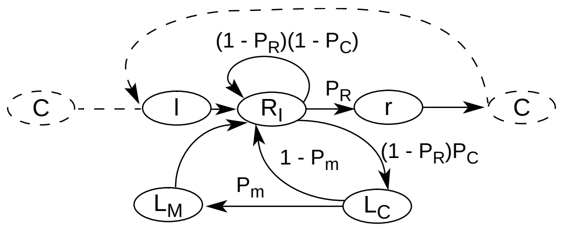


Figure A.2: Markov Chain of $R_{(2)}$. Message Passing Architecture without Prefetching

Execution of the activated thread resumes from the state l, where it loads the arrived data to a register. Then the thread is executed during a run interval R_I. If the run interval ends by the local access with the probability $(1-P_R)$, the thread either performs a cache access (L_C) with probability P_C or returns back to R_I (probability $(1-P_C)$). In case of a cache hit (probability $(1-P_m)$) the thread continues to run (R_I). On a cache miss with probability P_m the thread performs a local memory access (L_M), returning back to the R_I state. An instruction with remote reference (r) occurs with the probability P_R and causes a context switch (C).

Assume that the Markov chain is closed (state r directly connected to state l). To define the run length, R, the following system of equations must be solved with assumption $n_l = 1$:

$$\begin{cases} n_l = n_r = 1 \\ n_{R_I} = n_l + n_{L_M} + n_{L_C}(1-P_m) + n_{R_I}(1-P_R)(1-P_C) \\ n_{L_M} = n_{L_C}P_m \\ n_{L_C} = n_{R_I}(1-P_R)P_C \\ n_r = n_{R_I}P_R \\ R_{(2)} = R_I n_{R_I} + L_C n_{L_C} + L_M n_{L_M} + l n_l + r n_r \end{cases}$$

Thus the mean run length of a thread executed on a message passing architecture without data prefetching is:

$$R_{(2)} = \frac{R_I + (L_C + L_M P_m)(1-P_R)P_C}{P_R} + r + l \qquad \text{(A2)}$$

where P_R is a context switch probability.

A.3: Run Length in a Message Passing Architecture with Data Prefetching

Consider a switch-on-prefetch multi-threaded processor with strategy of context switching on explicit remote references (message passing architecture) with explicit data prefetching. The processors switches a context after each prefetch operation. A thread is specified by R_I, P_R, P_C, P_m, r and l. The mean number of remote data requested by a prefetch operation is m. The portion of remote data which can be prefetched is specified by a prefetch probability P_P.

Figure A.3(a) illustrates the Markov chain representing state transitions of the thread during a run length interval. The state mr represents a prefetch operation, where $m \geq 1$ prefetch requests are sent to the network. We assume that the PE performs an explicit context switch after prefetch operation, and a thread becomes suspended until all requested data arrives. The next prefetch operation is issued when the thread passes m remote references which was prefetched by previous prefetch operations. We assume that the probability of executing a prefetch operation after prefetched remote references is $1/m$.

Being reactivated after suspension (C) the thread is executed (R_I, L_C, L_M) until it needs remote data (P_R). If data is not prefetched (probability $(1-P_P)$) the thread initiates a remote access (r) and becomes suspended. If a prefetch was initiated the thread either continues to run (probability $P_P(1-1/m)$) or repeats the prefetch operation mr (probability P_P/m). If data was not prefetched (probability $(1-P_P)$) then the thread initiates a remote access (r) and becomes suspended.

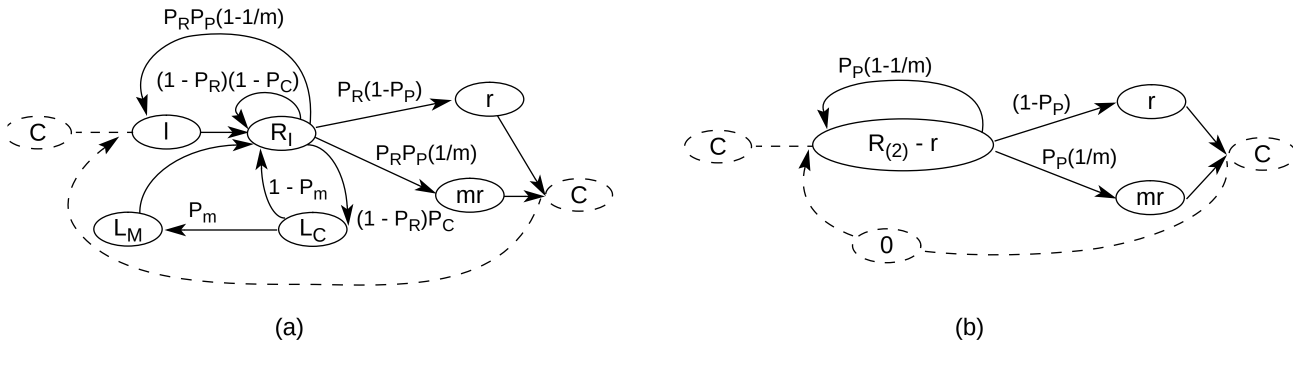


Figure A.3: Markov Chains of $R_{(3)}$. Message Passing Architecture with Prefetching

The closed Markov chain in Fig.A.3(a) can be transformed into the chain depicted in Fig.A.3(b), where $R_{(2)}$ is defined as (A2) and the state marked 0 is a dummy state. An expression for the mean run length of a thread executed on a message passing architecture with data prefetching can be derived using the same approach as in Section 1.2; the solution is:

$$R_{(3)} = \frac{mR_{(2)}}{m - P_P(m-1)} \tag{A3}$$

where $P_R(1 - P_P(1 - 1/m))$ is a context switch probability. Note, if $m = 1$ or $P_P = 0$ then $R_{(3)} = R_{(2)}$.

A.4: Run Length in a Shared Memory Architecture

Consider a switch-on-miss multi-threaded processor with context switching on a local memory miss (shared memory architecture). A thread is specified by R_I, l, probability of a cache access P_C. In this case a run interval R_I is the number of cycles between two consecutive memory accesses, cache (probability P_C) or registers ($1 - P_C$). The probability of a cache miss is P_m and the probability of a local memory miss is P_M. T is the duration of a locality test which is performed to check if a data resides in local or remote memory.

A Markov chain representing thread execution during a run length interval is depicted in Fig.A.4. On a cache miss with probability P_m the thread checks local memory (locality test T) and in the case of a local memory miss with probability P_M it becomes suspended.

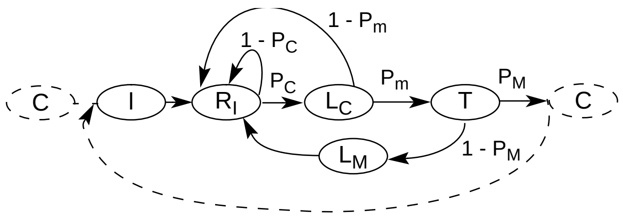


Figure A.4: Markov Chain of $R_{(4)}$. Shared Memory Architecture

The mean run length of a thread with context switching on a local memory miss (shared memory architecture) is:

$$R_{(4)} = \frac{R_I + (L_C + (T + L_M(1 - P_M))P_m)P_C}{P_C P_m P_M} + l \tag{A4}$$

where the product $P_C P_m P_M$ is the context switch probability.

A.5: Run Length in a Message Passing / Shared Memory Architecture

Assume that a multithreaded processor supports both shared memory and message passing programming models. A context switching policy aims to hide communication latency caused by explicit remote references in threads and local memory (cache) misses. The mean run length (A5) of a thread executed in an architecture with combination of shared memory and message passing mechanisms is derived from a Markov chain illustrated in Fig.A.5.

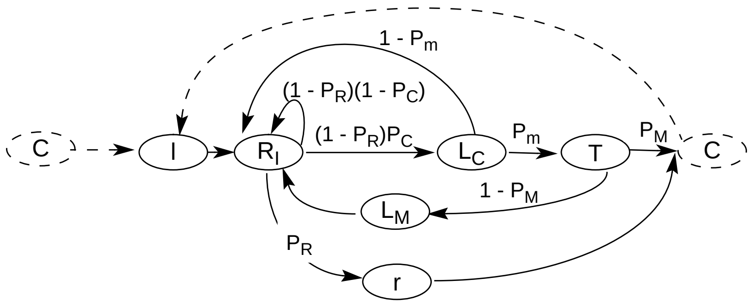


Figure A.5: Markov Chain of $R_{(5)}$. Message Passing / Shared Memory Architecture

$$R_{(5)} = l + \frac{R_I + (L_C + (T + L_M(1 - P_M))P_m)(1 - P_R)P_C + rP_R}{P_C P_m P_M(1 - P_R) + P_R} \tag{A5}$$

where the product $P_C P_m P_M(1 - P_R) + P_R$ is the context switch probability.

Appendix B:First-Order Approximation for MTA Efficiency

This Appendix summarizes results for the first-order approximation for efficiency of multi-threaded architecture (MTA). This approximation is valid for constant values of timing parameters. The basic assumptions are:

- An MTA executes a set of *n* identical threads (regularity).
- When a thread is suspended, all other (*n* - 1) threads pass through processor (linearity).

Expressions include the following parameters:

L = communication latency

C = context switch overhead

R = run length (a number of cycles between two consecutive remote references)

m = prefetch packet length (a number of prefetch requests generated by one prefetch operation)

P_P = prefetching probability (a portion of data which can be prefetched).

The MTA efficiency is denoted by the following variables:

E_1 is the efficiency of a single-threaded architecture.

E_n is the efficiency of an multi-threaded architecture.

E_L is the efficiency of the MTA on the linear region of efficiency curve, when number of threads is less then saturation point.

E_S is the efficiency of the MTA in saturation.

N_S is the saturation point of non-prefetching MTA and the reliable saturation point of prefetching MTA (minimum number of threads required to achieve saturation).

n_S is an optimistic saturation point of prefetching MTA.

The efficiency of non-prefetching MTA [12]:

$$\boxed{\begin{aligned} E_1 &= R/(R+L) \\ E_n &= \min(E_L, E_S) \\ E_L &= nR/(R+L+C) \\ E_S &= R/(R+C) \\ N_S &= L/(R+C)+1 \end{aligned}} \tag{B1}$$

The efficiency of prefetching MTA based on switch-on-prefetch model:

$$\boxed{\begin{aligned} &\text{Prefetching factor:}\quad \alpha_C = m/(m - mP_P + P_P) \\ &\text{Mean run length:}\quad R_C = \alpha_C R \\ &E_{1_C} = R_C/(R_C+L) \\ &E_{n_C} = \min(E_{L_C}, E_{S_C}) \\ &E_{L_C} = nR_C/(R_C+L+C) \\ &E_{S_C} = R_C/(R_C+C) \\ &n_{S_C} = L/(R_C+C)+1 \\ &N_S = L/(R+C)+1 \end{aligned}} \tag{B2}$$

The efficiency of prefetching MTA based on run-on-prefetch model:

$$\boxed{\begin{aligned}
&\text{Prefetching factor: } \alpha_R = m/(m - mP_P + P_P^2)\\
&\text{Overlapping factor: } \beta = P_P^2/(m - mP_P + P_P^2)\\
&\text{Mean run length: } R_R = \alpha_R R\\
&\text{Mean latency (thread idle time): } L_R = L - \beta R\\
&E_{1_R} = R_R/(R_R + L_R)\\
&E_{n_R} = \min(E_{L_R}, E_{S_R})\\
&E_{L_R} = nR_R/(R_R + C + L_R)\\
&E_{S_R} = R_R/(R_R + C)\\
&n_{S_R} = L_R/(R_R + C) + 1\\
&N_S = L/(R + C) + 1
\end{aligned}} \qquad \text{(B3)}$$

Figure B.2 - Figure B.6 depict different dependencies and comparisons of the first-order approximation for efficiency of prefetching and non-prefetching MTAs for the following set of parameters: $L = 128$, $C = 2$, $R = 16$.

B.1: Saturation of Prefetching MTA by Increasing Prefetching Factor of Its Work-Load

The optimistic saturation point, n_S, and the reliable saturation point, N_S, of prefetching MTA is defined by (B2) or (B3).

Plots $E(n, m)$ in Fig.B.2.b and Fig.B.3.b, as well as plots $E(n, P_P)$ in Fig.B.2.d and Fig.B.3.d show that in some cases the non-saturated MTA can be saturated by increasing prefetching factor in executed threads (i.e., the length of prefetch packets, m or/and the portion of prefetched data, P_P). The minimum value of the prefetch packet length, m_S (or prefetching probability), required to achieve MTA saturation is defined from:

$$\boxed{\begin{cases} E_L(m_S) = E_S(m_S) \\ m_S > 0 \end{cases} \qquad \begin{cases} E_L(P_S) = E_S(P_S) \\ 0 < P_S \le 1 \end{cases}} \qquad \text{(B4)}$$

In particular, for switch-on-prefetch model the minimum value of prefetching factor, α_{S_C}, required to achieve saturation is derived from $E_{L_C}(\alpha_{S_C}) = E_{S_C}(\alpha_{S_C})$:

$$\alpha_{S_C} = \left.\frac{L - C(n-1)}{R(n-1)}\right| \begin{aligned} m_S &= \frac{\alpha_{S_C} P_P}{1 - \alpha_{S_C}(1 - P_P)} > 0 \\ P_S &= \frac{m(\alpha_{S_C} - 1)}{(m-1)\alpha_{S_C}} \le 1 \end{aligned} \Bigg\} \qquad \text{(B5)}$$

If α_{S_C} can not be found as (B4), it means that the MTA can not be saturated by increasing α_C. In the latter case saturation can be achieved by loading more number of threads.

For example, plots $E(n, m)$ in Fig.B.2.b and Table 2 demonstrate that an MTA executing $n = 4$ (5, 8) threads is saturated when the length of prefetch packets achieves the value $m_4 = \lceil 14.96 \rceil$ ($m_5 = \lceil 3.55 \rceil$, $m_8 = \lceil 1.03 \rceil$) respectively. Thus, the larger the number of threads is , the smaller the length of prefetch packet is required to saturate the MTA. In case $n = 2$ or $n = 3$ the MTA can not be saturated by increasing m.

Plots $E(n, P_P)$ in Fig.B.2.d and Table 4 demonstrates that an MTA executing $n = 3(4, 5, 8)$ threads is saturated when the portion of prefetched data (prefetching probability, P_P) achieves a value $P_3 = 0.9892$ ($P_4 = 0.8087$, $P_5 = 0.6222$, $P_8 = 0.0234$), respectively. The larger the number of threads is, the smaller portion of prefetched data is required to saturate the MTA. In the case $n = 2$ the MTA can not be saturated by increasing P_P.

Plots $E(n, m)$ in Fig.B.3.b and Table 6 demonstrate that an MTA executing $n = 4$ (5, 8) threads is saturated when the length of prefetch packets achieves the value $m_4 = \lceil 8.45 \rceil$ ($m_5 = 2$, $m_8 = \lceil 0.57 \rceil$) respectively. In case $n = 2$ or $n = 3$ the MTA can not be saturated by increasing m.

Plots *Yen*, P_P) in Fig.B.3.d and Table 8 demonstrate that an MTA executing $n = 3(4, 5, 8)$ threads is saturated when the portion of prefetched data (prefetching probability, P_P) achieves a value 0.9304 (0.7188, 0.5268, 0.0176), respectively. In the case $n = 2$ the MTA can not be saturated by increasing P_P.

B.2: Limits of the MTA Efficiency

Limits of efficiency (B2) of switch-on-prefetch MTA:

$$\boxed{\left.\begin{aligned}
\lim_{P_P \to 1} \alpha_C &= m \\
\lim_{P_P \to 1} R_C &= mR \\
\lim_{P_P \to 1} E_{1_C} &= mR/(mR+L) \\
\lim_{P_P \to 1} E_{n_C} &= \min\left(\lim_{P_P \to 1} E_{L_C}, \lim_{P_P \to 1} E_{S_C}\right) \\
\lim_{P_P \to 1} E_{L_C} &= nmR/(mR+L+C) \\
\lim_{P_P \to 1} E_{S_C} &= mR/(mR+C) \\
\lim_{P_P \to 1} n_{S_C} &= L/(mR+C)+1
\end{aligned}\right\} \text{when } (P_P \to 1) \text{ and } (m \neq \infty)} \qquad \text{(B6)}$$

$$\boxed{\left.\begin{aligned}
\lim_{m \to \infty} \alpha_C &= 1/(1-P_P) \\
\lim_{m \to \infty} R_C &= R/(1-P_P) \\
\lim_{m \to \infty} E_{1_C} &= \frac{R}{R+L(1-P_P)} \\
\lim_{P_P \to 1} E_{n_C} &= \min\left(\lim_{m \to \infty} E_{L_C}, \lim_{m \to \infty} E_{S_C}\right) \\
\lim_{m \to \infty} E_{L_C} &= \frac{nR}{R+(L+C)(1-P_P)} \\
\lim_{m \to \infty} E_{S_C} &= \frac{R}{R+C(1-P_P)} \\
\lim_{m \to \infty} n_{S_C} &= \frac{L(1-P_P)}{R+C(1-P_P)}+1
\end{aligned}\right\} \text{when } (m \to \infty) \text{ and } (P_P < 1)} \qquad \text{(B7)}$$

Limits of efficiency (B3) of run-on-prefetch MTA:

$$\boxed{\left.\begin{aligned}
\lim_{P_P \to 1} \alpha_R &= m \\
\lim_{P_P \to 1} R_R &= mR \\
\lim_{P_P \to 1} E_{1_R} &= \frac{mR}{(m-1)R+L} \\
\lim_{P_P \to 1} E_{n_R} &= \min\left(\lim_{P_P \to 1} E_{L_R}, \lim_{P_P \to 1} E_{S_R}\right) \\
\lim_{P_P \to 1} E_{L_R} &= \frac{nmR}{(m-1)R+L+C} \\
\lim_{P_P \to 1} E_{S_R} &= mR/(mR+C) \\
\lim_{P_P \to 1} n_{S_R} &= \frac{L-R}{mR+C}+1
\end{aligned}\right\} \text{when } (P_P \to 1) \text{ and } (m \neq \infty)} \qquad \text{(B8)}$$

Limits of efficiency (B3) of run-on-prefetch MTA when $(m \to \infty)$ and $(P_P < 1)$ are the same as for switch-on-prefetch MTA (B6).

The length of prefetch packets m_r required to achieve $\gamma \cdot (100\%)$ of efficiency limit can be obtain from the following set of equations:

$$\boxed{\begin{array}{l} \begin{cases} E_L(m_1) = E_r \\ E_S(m_2) = E_r \\ m_r = \begin{cases} m_1, & \text{if } \min\{E_r, E_S(m_1)\} = E_r \\ m_2, & \text{if } \min\{E_L(m_2), E_r\} = E_r \end{cases} \end{cases} \\ \text{where } E_r = \gamma \cdot \min\left(\lim\limits_{m \to \infty} E_L(m), \lim\limits_{m \to \infty} E_S(m)\right) \end{array}} \tag{B9}$$

For example, assume that switch-on-prefetch MTA executes a set of threads (n = 5) with parameters L = 128, C = 2 and R = 16. A portion of remote references which can be prefetched is 65% (P_P = 0.65). Define a length of prefetch packets m_r required to achieve 99% of efficiency limit ($\gamma = 0.99$). The efficiency limit when $m \to \infty$ is 0.9581 (see (B7)) and 99% of efficiency limit is 0.9495.

The solution of (B9) is $m_r = \lceil 7.7066 \rceil = 8$ (m_1 = 3.6991; m_2 = 7.7066). Figure B.1 illustrate obtained solution. The curve $E_{L_C}(m)$ crosses the line E_r when $m = m_1$, however the MTA efficiency at this point is $E_{S_C}(m_1) < E_r$ (MTA is saturated). The curve $E_{S_C}(m)$ crosses E_r when $m = m_2$ and the MTA efficiency is $E_{S_C}(m_2) = E_r$. Thus $m_r = \lceil m_2 \rceil$.

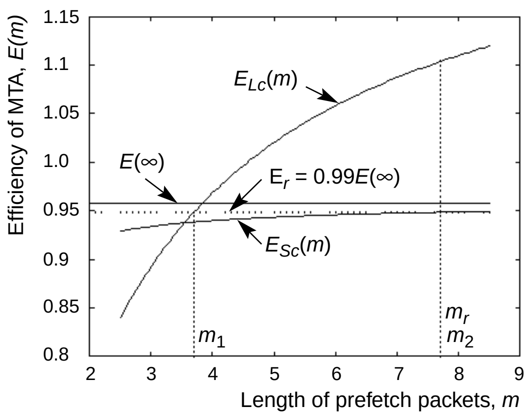
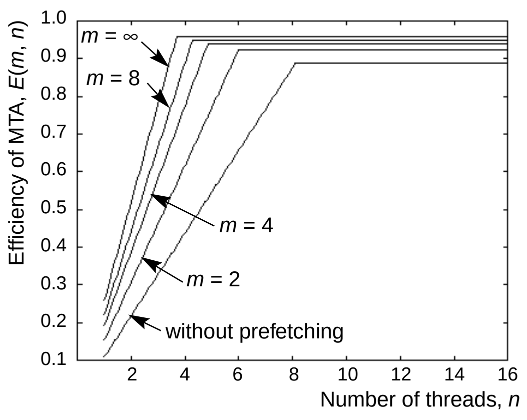
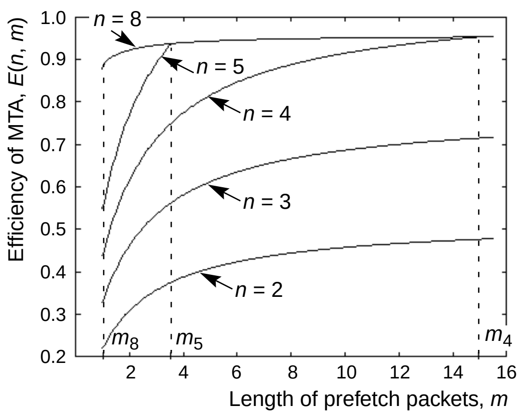


Figure B.1: Length of Prefetch Packet Required To Achieve 99% of Efficiency Limit

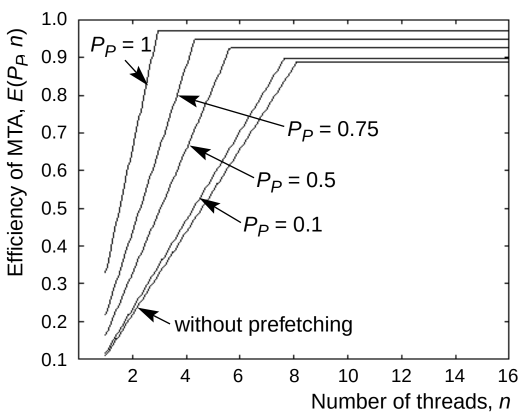
This example demonstrates that if even prefetching MTA is saturated, its efficiency can be improved by increasing the prefetching factor of its load.



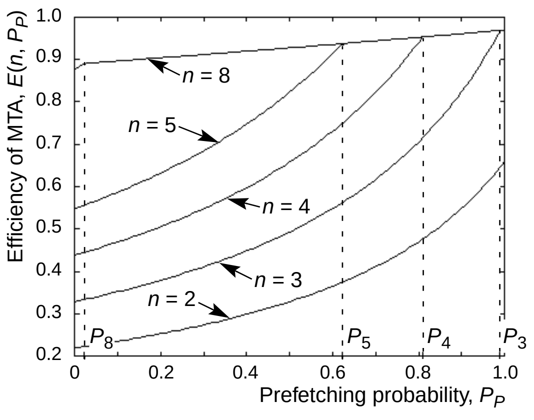
(a) as a function of m and n ($P_P = 0.65$)



(b) as a function of n and m ($P_P = 0.65$)



(c) as a function of P_P and n ($m = 4$)



(d) as a function of P_P and n ($m = 4$)

Figure B.2: Switch-On-Prefetch: MTA Efficiency (*R* = 16, *L* = 128, *C* = 2)

TABLE 1. Saturation Points, n_S, and Efficiency in Saturation (Fig.B.2.a)

m	0*	2	4	8	bb
α_C	-	1.48	1.95	2.32	2.86
n_S	8.11	5.98	4.85	4.27	3.68
E_S	0.8889	0.9222	0.9398	0.9489	0.9581

* Non-prefetching MTA

TABLE 2. Saturation Points, m_S (Fig.B.2.b)

n	2	3	4	5	8
m_S	-*	-*	14.98	3.55	1.03

* The MTA can not be saturated by increasing m

Figure B.3: Run-On-Prefetch: MTA Efficiency (R =

TABLE 3. Saturation Points, n_S, and Efficiency in Saturation (Fig.B.2.c)

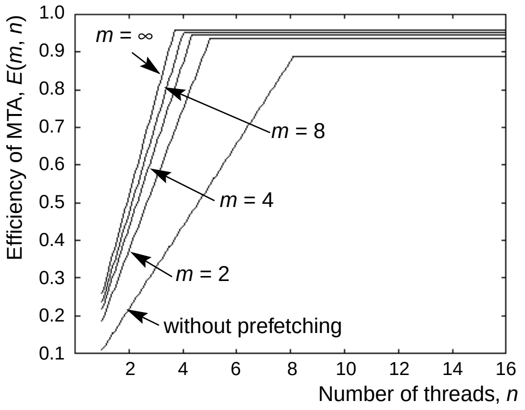
P_P	0*	0.1	0.5	0.75	1
α_C	-	1.08	1.6	2.29	4
n_S	8.11	7.63	5.64	4.32	2.94
E_S	0.8889	0.8964	0.9275	0.9481	0.9697

* Non-prefetching MTA

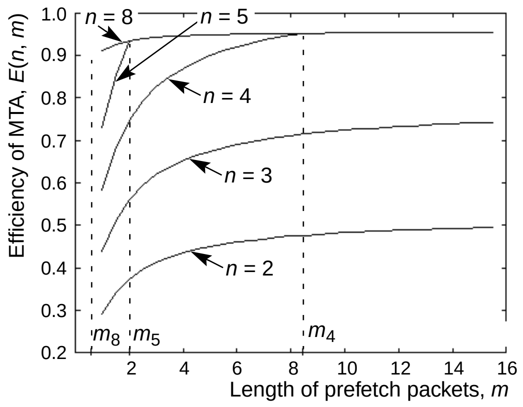
TABLE 4. Saturation Points, P_S (Fig.B.2.d)

n	2	3	4	5	8
P_S	-*	0.9892	0.8087	0.6222	0.0234

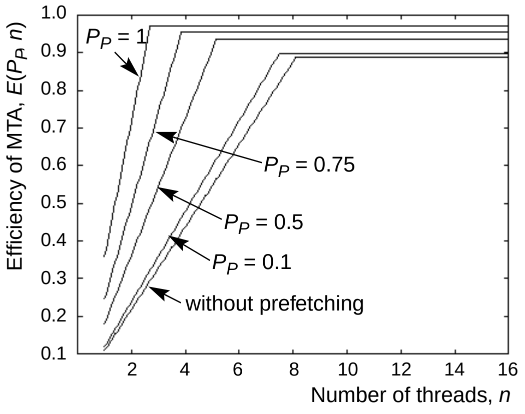
* The MTA can not be saturated by increasing P_P



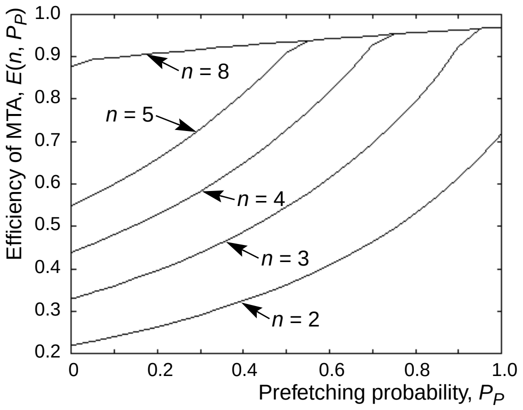
(a) as a function of m and n ($P_P = 0.65$)



(b) as a function of n and m ($P_P = 0.65$)



(c) as a function of P_P and n ($m = 4$)



(d) as a function of P_P and n ($m = 4$)

16, L = 128, C = 2)

TABLE 5. Saturation Points, n_S, and Efficiency in Saturation (Fig.B.3.a)

m	0*	2	4	8	bb
α_C	-	1.78	2.205	2.48	2.86
β_C	-	0.38	0.23	0.13	0
n_S	8.11	5.00	4.35	4.02	3.68
E_S	0.8889	0.9344	0.9461	0.9521	0.9581

* Non-prefetching MTA

TABLE 7. Saturation Points, n_S, and Efficiency in Saturation (Fig.B.3.c)

P_P	0*	0.1	0.5	0.75	1
α_C	-	1.11	1.78	2.56	4
β_C	-	0.003	0.11	0.36	1
n_S	8.11	7.49	5.20	3.98	2.94
E_S	0.8889	0.8986	0.9343	0.9534	0.9697

* Non-prefetching MTA

TABLE 6. Saturation Points, m_S (Fig.B.3.b)

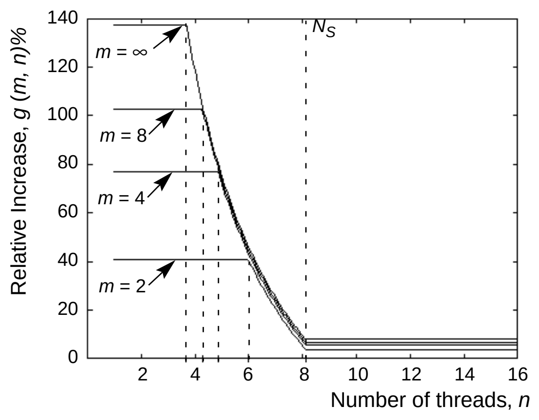
n	2	3	4	5	8
m_S	-*	-*	8.45	2.00	0.57

* The MTA can not be saturated by increasing m

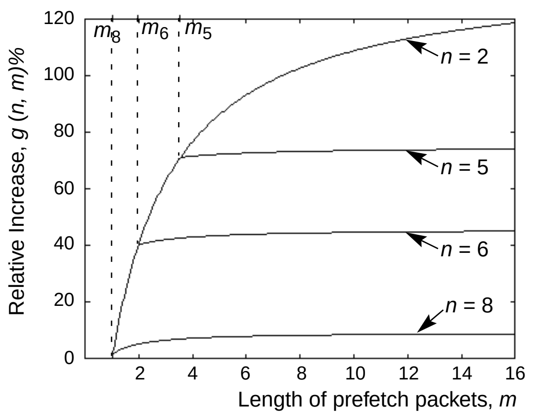
TABLE 8. Saturation Points, P_S (Fig.B.3.d)

n	2	3	4	5	8
P_S	-*	0.9304	0.7188	0.5268	0.0176

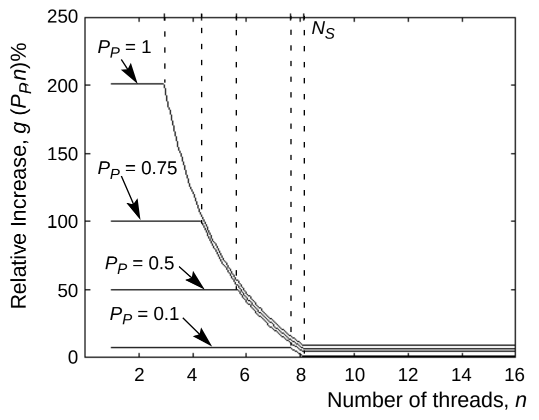
* The MTA can not be saturated by increasing P_P



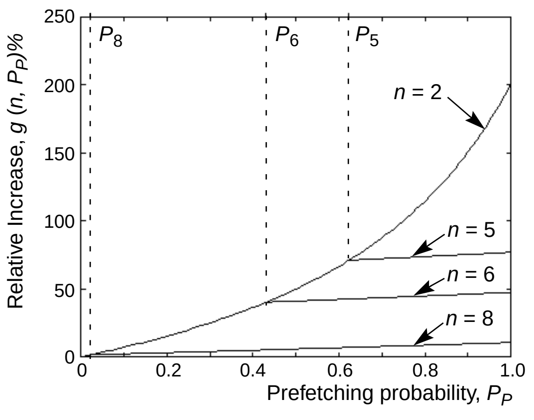
(a) as a function of m and n (P_P = 0.65)



(b) as a function of n and m (P_P = 0.65)



(c) as a function of P_P and n (m = 4)



(d) as a function of P_P and n (m = 4)

Figure B.4: Relative Increase in Efficiency of Switch-On-Prefetch MTA vs. Non-Prefetching MTA (R = 16, L = 128, C = 2)

TABLE 9. Saturation Points, n_S (Fig.B.4.a)

m	n_{S_C}	n_S	g(%) at n_{S_C}	g(%) at n_S
2	5.98	8.11	40.72	3.75
4	4.85		76.70	5.72
8	4.27		102.60	6.75
∞	3.68		137.40	7.78

TABLE 11. Saturation Points, n_S (Fig.B.4.c)

P_P	n_{S_C}	n_S	g(%) at n_{S_C}	g(%) at n_S
0.1	7.63	8.11	7.16	0.84
0.5	5.64		50.13	4.35
0.75	4.32		100.34	6.67
1	2.94		201.03	9.09

TABLE 10. Saturation Points, m_S (Fig.B.4.b)

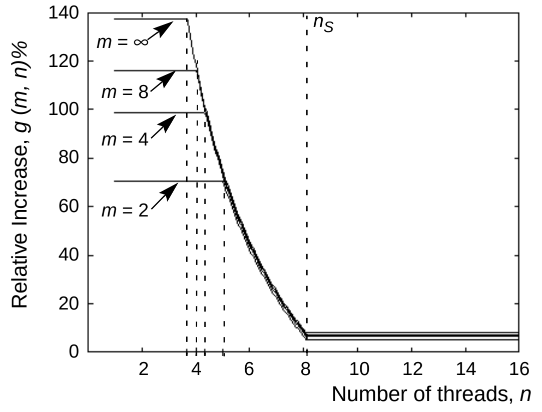
n	m_S	g(%) at m_S	g(%) at $m = \infty$
2	-*	76.70†	137.40
5	3.55	71.09	74.85
6	1.98	40.20	45.71
8	1.03	1.59	9.28

* The prefetching MTA can not be saturated by increasing m
† At the point m = 4

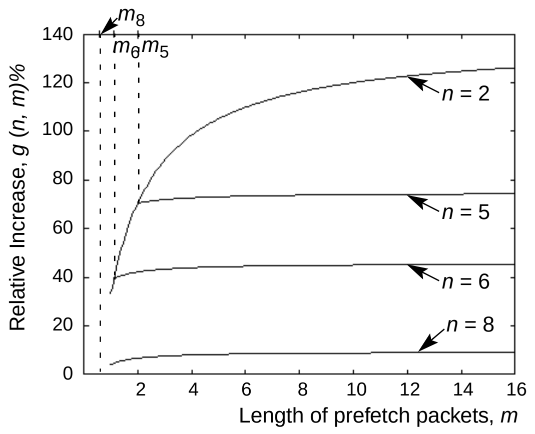
TABLE 12. Saturation Points, P_S (Fig.B.4.d)

n	P_S	g(%) at P_S	g(%) at P_P = 1
2	-*	71.09†	201.03
5	0.62	71.09	76.97
6	0.43	40.20	47.48
8	0.02	1.59	10.61

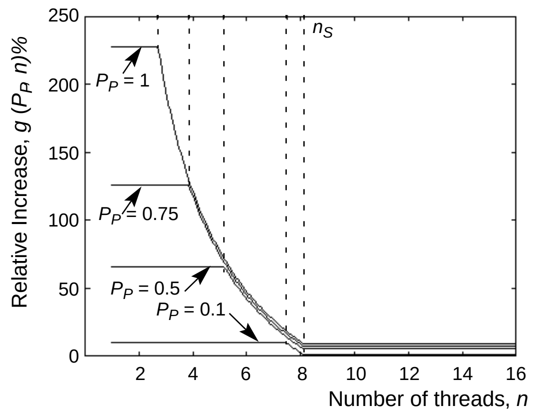
* The prefetching MTA can not be saturated by increasing P_P
† At the point P_P = 0.62



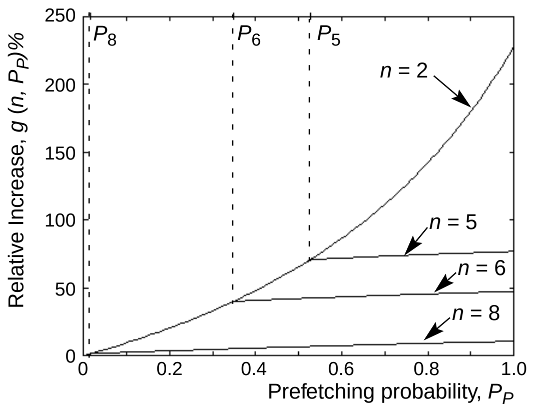
(a) as a function of m and n (P_P = 0.65)



(c) as a function of n and m (P_P = 0.65)



(b) as a function of P_P and n (m = 4)



(d) as a function of P_P and n (m = 4)

Figure B.5: Relative Increase in Efficiency of Run-On-Prefetch MTA vs. Non-Prefetching MTA (R = 16, L = 128, C = 2)

TABLE 13. Saturation Points, n_S (Fig.B.5.a)

m	n_{S_C}	n_S	g(%) at n_{S_C}	g(%) at n_S
2	5.00	8.11	70.60	5.13
4	4.35		98.53	6.44
8	4.02		116.23	7.10
∞	3.68		137.40	7.78

TABLE 15. Saturation Points, n_S (Fig.B.5.c)

P_P	n_{S_C}	n_S	g(%) at n_{S_C}	g(%) at n_S
0.1	7.49	8.11	9.54	1.10
0.5	5.15		65.67	5.11
0.75	3.85		126.24	7.26
1	2.70		228.09	9.09

TABLE 14. Saturation Points, m_S (Fig.B.5.b)

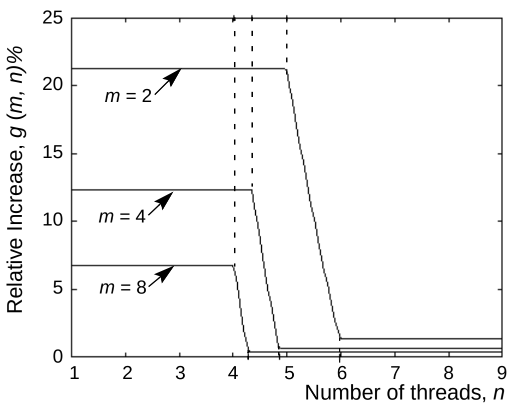
n	m_S	g(%) at m_S	g(%) at $m = \infty$
2	-*	98.53†	137.40
5	2.00	70.53	74.85
6	1.11	39.38	45.71
8	0.57	4.02‡	9.28

* The prefetching MTA can not be saturated by increasing m
† At the point m = 4
‡ At the point m = 1

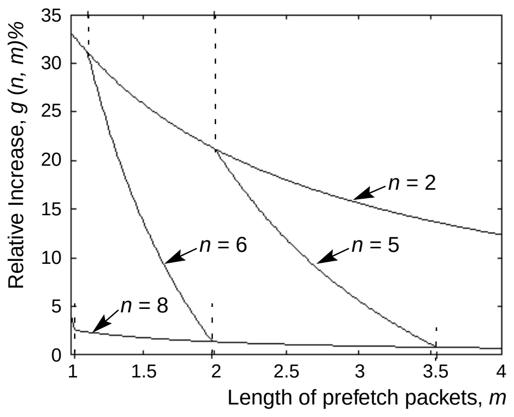
TABLE 16. Saturation Points, P_S (Fig.B.5.d)

n	P_S	g(%) at P_S	g(%) at P_P = 1
2	-*	70.91†	228.09
5	0.53	70.91	76.97
6	0.35	40.14	47.48
8	0.02	1.59	10.61

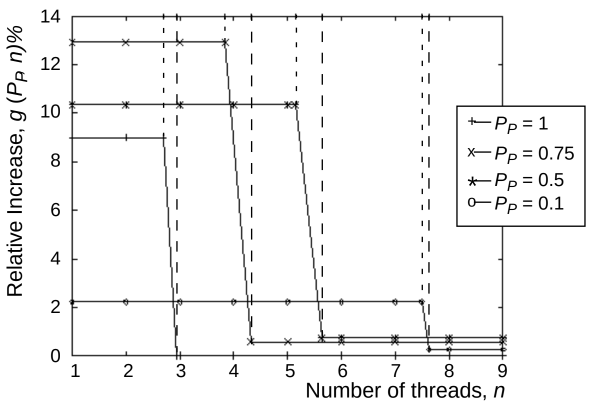
* The prefetching MTA can not be saturated by increasing P_P
† At the point P_P = 0.53



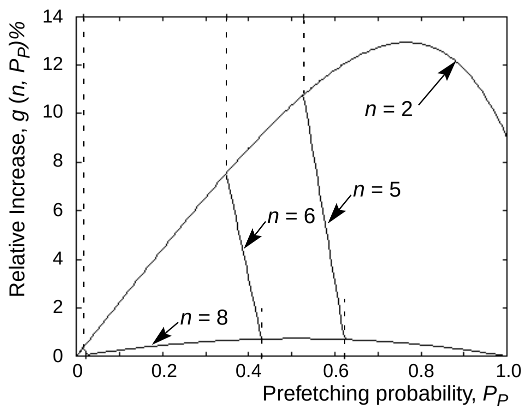
(a) as a function of *m* and *n* (P_P = 0.65)



(b) as a function of *n* and *m* (P_P = 0.65)



(c) as a function of P_P and *n* (*m* = 4)



(d) as a function of P_P and *n* (*m* = 4)

Figure B.6: Relative Increase in Efficiency of Run-On-Prefetch MTA vs. Switch-On-Prefetch MTA (*R* = 16, *L* = 128, *C* = 2)

TABLE 17. Saturation Points, n_S (Fig.B.6.a)

m	n_{S_R}	n_{S_C}	g(%) at n_{S_R}	g(%) at n_{S_C}
2	5.00	5.98	21.23	1.33
4	4.35	4.85	12.35	0.67
8	4.02	4.27	6.73	0.34

TABLE 18. Saturation Points, m_S (Fig.B.6.b)

n	m_{S_R}	m_{S_C}	g(%) at m_{S_R}	g(%) at m_{S_C}
2	-*	-*	12.35†	6.73‡
5	2.00	3.55	21.25	0.76
6	1.11	1.98	31.15	1.34
8	0.57	1.03	4.02**	2.53

* The MTA can not be saturated by increasing *m*
† At the point *m* = 4
‡ At the point *m* = 8
** At the point *m* = 1

TABLE 19. Saturation Points, n_S (Fig.B.6.c)

P_P	n_{S_R}	n_{S_C}	g(%) at n_{S_R}	g(%) at n_{S_C}
0.1	7.49	7.63	2.23	0.25
0.5	5.15	5.64	10.36	0.73
0.75	3.85	4.32	12.93	0.56
1	2.70	2.94	8.99	0

TABLE 20. Saturation Points, P_S (Fig.B.6.d)

n	P_{S_R}	P_{S_C}	g(%) at P_{S_R}	g(%) at P_{S_C}
2	-*	-*	10.78†	12.07‡
5	0.53	0.62	10.78	0.69
6	0.35	0.43	7.55	0.71
8	0.02	0.02	0.39	0.06

* The MTA can not be saturated by increasing P_P

† At the point P_P = 0.53

‡ At the point P_P = 0.62

References

[1] A. Agarwal, “Performance Tredeoffs in Multithreaded Processors”, *IEEE Transactions on Parallel and Distributed Systems*, 3(5): 525-539, September 1992.

[2] F. Baskett, K.M. Chandy, R.R.Muntz and F.G. Palacios, “Open, Closed, and Mixed Networks of Queues with Different Classes of Customers“, *Journal of the ACM*, 22(2): 248-260, April 1975.

[3] B. Boothe and A. Ranade, “Improved Multithreading Techniques for Hiding Communication Latency in Multiprocessors”, in *Proc. 19th Ann. Int. Symp. on Comp. Arch.*, pp. 241-223, 1992.

[4] P. Cao, E.W. Felten, a.R. Karlin and K. Li, “Implementation and Performance of Integrated Application-Controlled Caching, Prefetching and Disk Scheduling“, Tech. Rep. CS-TR-493-95, Dept. of Comp. Sci., Princeton University, Princeton, NJ, 1995.

[5] T.-F. Chen and J.-L. Baer, “Reducing Memory Latency via Non-blocking and Prefetching Caches“, in *Proc. of the 5th Int. Conf. on Architectural Support for Programming Languages and Operation Systems*, pp. 51-61, 1992.

[6] T.-F. Chen, “Data Prefetching for High-Performance Processors“, Ph.D. dissertation, UW-CSE-93-07-01, Dept. of Comp. Sci. and Engineering, University of Washington, Seattle WA, July 1993.

[7] A. Gupta, J. Henessy, K. Gharachorloo, T. Mowry, and W.-D. Weber, “Comparative Evaluation of Latency Reducing and Tolerating Techniques“, in *Proc. of the 18th Annual Int. Symp. on Computer Arch.*, pp. 254-533, 1991.

[8] R. Jain, “The Art of Computer Systems Performance Analysis”, John Wiley & Sons, Inc.,1991.

[9] T.C. Mowry, M.S. Lam and A. Gupta, “Design and Evaluation of a Compiler Algorithm for Prefetching“, in *Proc. of the 5th Int. Conf. on Architectural Support for Programming Languages and Operation Systems*, pp. 62-73, 1992.

[10] S. S. Nemawarkar, R. Govindarajan, G.R. Gao and V.K. Agarwal, “Analysis of Multithreaded Multiprocessors with Distributed Shared Memory”, in *Proc. of the 5th Int. Symp. on Parallel and Distr. Processing (SPDP)*, 1993.

[11] M. Reiser and S.S. Lavenberg, “Mean-Value Analysis of Closed Multichain Queuing Networks“, *Journal of the ACM*, 27(2): 313-322, April 1980.

[12] R. H. Saavedra-Barrera, D. E. Culler, and T. von Eicken, “Analysis of Multithreaded Architectures for Parallel Computing“, in *Proc. of the 2nd Ann. ACM Symp. on Parallel Algorithms and Architectures*, pp. 169-178, 1990.

[13] R. H. Saavedra-Barrera and D. E. Culler, “An Analytical Solution for a Markov Chain Modeling Multithreaded Execution“, Tech. Rep. UCB/CSD-91-623, University of California, Berkeley, 1991.

[14] V. Vlassov, H. Ahmed, L.-E. Thorelli and R. Ayani, “A Simulation Platform for Multi-Threaded Architectures“, in *Proc. of the 4th Int. Workshop on Modeling, Analysis and Simulation of Comp. and Telecom. Systems (MASCOTS)*, pp. 103-108, Feb 1996.