ABOUT NUMBER OF EFFECTIVELY PERFORMED STREAMS IN MULTI-CORE COMPUTERS WITH SHARED MEMORY

Nedzelskyi D.O.

The article investigates the dependence of the number of effectively executed programs in multi-core computers with shared memory on the parameters of programs and computers. All computer cores execute parallel streams of a single program developed in accordance with the OpenMP API. There are no interactions between streams program. Conflicts can occur only when cores are accessed into shared memory. Suggested the necessary models. Analytical expressions are obtained for the dependence of the number of effectively executed programs on the properties of programs, core and shared memory parameters. The main reason for limiting the number of effectively executed programs is the overload of the shared memory. The threshold value of the number of effectively executed programs has been determined.

**Keywords**: multi-core computer, core, shared memory, acceleration factor, efficiency, stream

1. Introduction

Efficient utilization of the cores in the computers with high number of cores is one of the most important problems for designers of software for these systems. Multi-core computers performance is defined using acceleration rate when executing an application with parallel streams or a set of applications without parallel streams when utilizing n cores. In reality, the acceleration depends on many factors.

In [1-7] multi-core computer utilization is obtained by testing/running real applications on the system. These experimental approaches do not generalize well and discovered accelerators are not always applicable or even partially applicable to the next application. Also the mentioned articles do not analyze the features which caused acceleration and the reasons for the limit of acceleration where an increase in number of cores does not increase the performance any longer.

The author is not aware of model analytical approaches to model/calculate efficient utilization (performance) of multi-core computers. Existing works in multi-core computer systems do not have an analytical approach estimating of the number of effectively used cores.

The purpose of the article is to study the efficiency of using cores (threads) in multi-core computers with shared memory depending on the number of cores, properties program performed by the cores, core and shared memory parameters when executing parallel programs developed in accordance with the OpenMP API.

2. Efficient utilization of a multi-core computer with n cores executing single parallel application

Efficient utilization of multi-core computer is calculated using the acceleration rate. The acceleration rate of a multi-core computer is defined as:

$$S = \frac{T_1}{T_n} = \frac{Pr_1}{Pr_n},$$

where $T_1$ – application execution time of a single core, $T_n$ – application execution time using $n$ cores where the application consists of exactly $n$ streams, $Pr_1$ - performance of a single core when there is no influence from other cores, $Pr_n$ - performance of an $n$ core computer.

Since all $n$ cores perform the same stream of a parallel application the core utilization rate is the same for all cores and utilization rate of the multi-core computer in this case equals the single core utilization rate $H_k = H_1$, where $H_1$ - single core utilization rate, when all $n$ cores are executing a parallel program; $H_k$ - multi-core computer utilization rate when all $n$ cores are executing a parallel program.

Multi-core computer performance where all $n$ cores are utilised is defined as...
Multi-core computer acceleration rate in this case equals
\[ S = \frac{Pr_n}{Pr_i} = \frac{Pr_i \cdot \sum_{i=1}^{n} H_i}{Pr_i} = n \cdot H_i. \]

3. Model of a multi-core computer with shared memory with all cores are busy
Preconditions:
- All cores are busy executing non-interacting streams of parallel applications. In other words, there are no interactions between executable code. The next instruction is executed by core straight away unless all of the following conditions exist:
  - It requires memory access.
  - The requested data is not present in the memory cache.
  - The cache is full. If all these conditions are exist the core is blocked and the instruction is not executed.
- Each core while executing its application generates data requests to the shared memory. Data requests from a single core is calculated according to the formula:
\[ \lambda_i = \frac{N_i \cdot \omega_i \cdot P_i}{N_i \cdot t_{INS}^i} = \frac{\omega_i \cdot P_i}{t_{INS}^i}, \]
where \( N_i \) - the number of instruction in the running program; \( \omega_i \cdot P_i \) - the probability of instructions to access to memory in the i-th program; \( t_{INS}^i \) - average time to execute one instruction in the core.
- Data requests from all cores to the shared memory are exponentially distributed.
- Memory is a compound device, where new requests are served only once the previous request has been completed.
- Served memory requests are also distributed according to an exponential distribution.
- Memory serves requests from all cores with intensity \( \mu_{MEM} = 1 / t_{MEM} \), where \( t_{MEM} \) - average time to execute one request by memory.
- Data requests to the memory are served using FIFO.

Multi-core computers can be described as a 2-phase system serving a high volume of requests. Phase 1 comprises of n cores executing applications and generating requests to the shared memory. Phase 2 occurs in shared memory itself. Buffers with capacity for k requests are placed between phases.

Due to difficulties in analytical representation of this general model this article explores a simplified equivalent version of the model (Fig.1).

Phase 1 of the equivalent model consists of an equivalent core. The equivalent core performance (frequency of generated requests) is assumed to be a sum of performances of all original model cores
\[ \lambda_{eqv} = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \frac{\omega_i P_i}{t_{INS}^i}. \]

The equivalent model has a single buffer and its size is the sum of the original model buffer size.
Phase 2 of the equivalent model is memory. Its performance (execution frequency) is equal to that of the original model. Buffer selection and core blocking logic are identical in both models.

Hence, from the performance point of view, the original and simplified models of a multi-core computer are equivalent.

The overall load rate of the shared memory for the n-core computer is calculated according to the formula
\[ \rho_{MEM}^x = \sum_{i=1}^{n} \frac{\lambda_i}{\mu_{MEM}} = \sum_{i=1}^{n} \frac{\omega_i P_i}{t_{INS}^i} = \sum_{i=1}^{n} \rho_i P_i. \]

where \( \rho_{MEM}^x \) - memory load rate from all cores; \( \rho_i P_i \) - memory load rate from i-th core.

Solving simultaneous equations for selected overall load rate ranges we obtain the following expressions for the model utilization rates:

**Equivalent core utilization rate:**
\[ H_k = 1 \] if \( \rho_{MEM}^x < 1; \) (2)
\[ H_k = \frac{k+1}{k+2} \] if \( \rho_{MEM}^x = 1; \) (3)
\[ H_k = \frac{1-(\rho_{MEM}^x)^{k+1}}{1-(\rho_{MEM}^x)^{k+2}} \] if \( \rho_{MEM}^x > 1; \) (4)

**Shared memory utilization rate:**
\[ E = \frac{\rho_{MEM}^x \cdot [1-(\rho_{MEM}^x)^{k+1}]}{1-(\rho_{MEM}^x)^{k+2}} \] if \( \rho_{MEM}^x < 1; \) (5)
\[ E = 1 \] if \( \rho_{MEM}^x > 1; \) (6)
\[ E = \frac{k+1}{k+2} \] if \( \rho_{MEM}^x = 1. \) (7)
\[ \rho_{\text{MEM}}^\Sigma = \sum_{i}^{n} \rho_{\text{MEM}}^i = 1. \]

Consider an example, displaying multi-core computer efficient utilization (acceleration rate) dependency on the number of cores utilised when executing a parallel application.

To simplify calculations, we assume an infinite buffer size between a core and the shared memory. The limit of core utilization rate with infinite buffer size for selected ranges of shared memory load rates is defined below:

\[ \lim_{n \to \infty} H_k = \frac{1}{\rho_{\text{MEM}}^\Sigma} \quad \text{if} \quad \rho_{\text{MEM}}^\Sigma > 1 \]
\[ \lim_{n \to \infty} H_k = 1 \quad \text{if} \quad \rho_{\text{MEM}}^\Sigma \leq 1. \]

Intermediate conclusion:

with \( n = 8 \), \( \rho_{\text{MEM}}^i = 0.125 \);
\[ \rho_{\text{MEM}}^\Sigma = 8 \times 0.125 = 1; \quad H_k = 1; \quad S = 8; \]

with \( n = 12 \), \( \rho_{\text{MEM}}^i = 0.125 \);
\[ \rho_{\text{MEM}}^\Sigma = 12 \times 0.125 = 1.5; \quad H_k = 2 / 3; \quad S = 8; \]

with \( n = 16 \), \( \rho_{\text{MEM}}^i = 0.125 \);
\[ \rho_{\text{MEM}}^\Sigma = 16 \times 0.125 = 2; \quad H_k = 0.5; \quad S = 8. \]

The example demonstrates that there is an upper limit (threshold) on the number of cores after which the performance of a multi-core computer does not improve.

This threshold (practical number of cores \( n \)) for parallel applications designed using OpenMP API defines the overall shared memory load rate generated by all cores of the multi-core computer.

\[ \rho_{\text{MEM}}^\Sigma = \sum_{i}^{n} \rho_{\text{MEM}}^i = 1. \]

The multi-core computer acceleration rate \( S \) while executing parallel applications can be calculated using defined above overall shared memory load rate:

\[ S = n \quad \text{when} \quad \rho_{\text{MEM}}^\Sigma \leq 1; \]
\[ S = \frac{n}{\rho_{\text{MEM}}^\Sigma} \quad \text{when} \quad \rho_{\text{MEM}}^\Sigma > 1. \]

Conclusion

1. The multi-core computer acceleration rate while executing parallel applications consisting of \( n \) streams is a product of the number of cores (application streams) and a single core utilization rate

\[ S = n \times H_k. \]

2. The practical number of streams of a parallel application defines the threshold of the overall shared memory load rate generated by all cores of the multi-core computer

\[ \rho_{\text{MEM}}^\Sigma = \sum_{i}^{n} \rho_{\text{MEM}}^i = 1. \]

3. The number of effectively used cores (streams in these cores) can be determined by the formulas

\[ n = S \quad \text{when} \quad \rho_{\text{MEM}}^\Sigma \leq 1; \]
\[ n = S \times \rho_{\text{MEM}}^\Sigma \quad \text{when} \quad \rho_{\text{MEM}}^\Sigma > 1. \]

4. The proposed methodology can be used to estimate from above the number of threads and, accordingly, the number of cores when developing parallel applications in accordance with the OpenMP API.

5. Shared memory overload is the main cause of a multi-core computer sublinear acceleration rate when executing parallel application using OpenMP API. The overload will cause the cores to be idle for a significant percentage of time.

References

Недзельський Д.О. Ефективність виконання потоків у багатоцільовому комп’ютері з загальною пам’ятю

У статті досліджується залежність кількості ефективно виконуваних програм в багатоядерних комп’ютерах із загальною пам’ятю від параметрів програм і комп’ютерів. Всі комп’ютерні ядра виконують паралельні потоки однієї програми, розробленої відповідно до API OpenMP. Там немає взаємодії між потоками програм. Конфлікти можуть виникати тільки при зверненні до ядер в загальній пам’яті. Запропоновано необхідні моделі. Отримано аналітичні вирази для залежності кількості ефективно виконуваних програм від властивостей програм, параметрів ядра та розділяється пам’яті. Визначено граничне значення кількості ефективно виконуваних програм.

**Ключові слова:** багатоядерний комп’ютер, ядро, колективна пам’ять, коекфіцієнт прискорення, ефективність, потік

Недзельський Д.А. О Ефективність виконуваних потоків в многоцільових комп’ютерах з обшою пам’ятю

В статье исследуется зависимость количества эффективно выполняемых программ в многоядерных компьютерах с общей памятью от параметров программ и компьютеров. Все компьютерные ядра выполняют параллельные потоки одной программы, разработанной в соответствии с API OpenMP. Там нет взаимодействия между потоками программы. Конфликты могут возникать только при обращении к ядрам в общей памяти. Предложены необходимые модели. Получены аналитические выражения для зависимости количества эффективно выполняемых программ от свойств программ, параметров ядра и разделяемой памяті. Ограничена зависимость количества эффективно выполняемых программ является перегрузка разделяемой памяті. Определено пороговое значение количества эффективно выполняемых программ.

**Ключевое слово:** многоядерный компьютер, ядро, разделяемая память, коэффициент ускорения, эффективность, поток

Недзельский Дмитро Олександрович – к.т.н., доцент, доцент кафедры компьютерных наук та инженерии Східноукраїнського національного університету імені Володимира Даля, e-mail: nedzelsky946@gmail.com

Стаття подана 16.08.2019.