# A Scheduling Algorithm for Asymmetric Processor Architecture

S.Subha
School of Information Technology
and Engineering
VIT, Vellore, India

## ABSTRACT
Chip multiprocessors are used widely today. The cores in a chip can be homogeneous or heterogeneous. This paper proposes a scheduling algorithm for heterogeneous multiprocessors wotj, multiple functional units of varying speed in each processor. Instructions that can be scheduled in parallel are considered. An optimization function is developed to allocate the processes to the processors that minimize the overall execution time. The proposed model is simulated for a chosen example and verified to give 46% improvement in performance.

## General Terms
Process scheduling

## Keywords
Chip multiprocessors, Process scheduling

## 1. INTRODUCTION
Chip multiprocessors can be homogeneous or heterogeneous. In homogeneous CMP, all the processors belong to the same family. They have the same speed. In heterogeneous CMP, processors belong to different families. Their processing speed differs. Each processor in CMP can have multiple functional units for the same operation. The speed of individual functional unit may vary. Any program segment can be visualized to start with a serial part following by <parallel part, serial part> pairs ending with a serial part. Scheduling the parallel part on symmetric (homogeneous) processors can be achieved by adopting any scheduling algorithm. Scheduling the parallel part on asymmetric (heterogeneous) processor is a tougher task as the processing speed differs. It is logical to schedule every type of instruction on a processor which executes in minimum time. However, in reality this may not be feasible based on the instruction mix.

The model in [1] describes scheduling of processes on asymmetric processors with dependencies. This paper considers processes that can be scheduled in parallel without data dependencies on CMP. It is also assumed that each processor in CMP has multiple functional units for each instruction type with varying speed. This paper proposes an optimization function that determines the schedule of instructions in the parallel part of any program segment in asymmetric CMP. The solution to this function gives the schedule.
The rest of the paper is as follows. Section 2 gives the motivation, section 3 the proposed model, section 4 simulation, section 5 conclusion, section 6 references.

## 2. MOTIVATION
Consider asymmetric CMP system with four processors belonging to f families. Let the parallel part of the program segment contain four load instructions, six store instructions, four multiply instructions, twenty floating point operations. The following table Table1 gives the execution time in units of time of each type of instruction on the four processors.

**Table 1**

| Processor#/instruction | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Load | 3 | 4 | 7 | 5 |
| Store | 2 | 3 | 6 | 3 |
| Multiply | 6 | 8 | 4 | 10 |
| Floating point | 15 | 20 | 13 | 30 |

From Table 1 it can be seen that it takes eight units to perform multiply operation on processor two, thirty units of time to perform floating point operation on processor four. The rest of the table can be interpreted in similar way.

The following schedule in Table 2 gives optimal scheduling

**Table 2**

| Processor#/instruction | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Load | 0 | 0 | 2 | 2 |
| Store | 6 | 0 | 0 | 0 |
| Multiply | 2 | 0 | 1 | 1 |
| Floating point | 7 | 6 | 3 | 4 |

The optimal time is 140 units of time. From Table 2 it can be seen that three floating point instructions are scheduled on processor three, two load instructions are scheduled on processor four etc. The optimal time for the schedule is given as follows. It takes 3*0 time units for Load instruction on processor one, 4*0 time units on processor two, 7*2=14 time units on processor three, 5*2=10 units of time on processor four for Load instruction. For Store instruction the time taken is 12 units of time on processor one and zero units of time on other processors. For multiply instruction, the time taken is 2*6=12 time units on processor one, zero time units on processor two, 1*4=4 time units on processor three, 1*10=10 time units on processor four. For floating point operation, the execution time is 105, 120, 39, 120 units on processors 1, 2, 3, 4 respectively. The total time taken is maximum time on any processor which is 140 time units on processor four.

The same instruction mix if allotted based on the following strategy gives the total execution time to be 260ns. Allot to the processor

that has minimum execution time. According to this rule, Load is allotted to processor-1 takes 4*5 = 20 units of time Store is allotted to processor-1 takes 6*2=12 units of time. Multiply is allotted to processor-3 takes 4*4 = 16 units of time. Floating point is allotted to processor-3 takes 20*13 = 260 units of time. Store needs to follow load on same processor.

**Table 3**

| Processor#/instruction | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Load | 4 | 0 | 0 | 0 |
| Store | 6 | 0 | 0 | 0 |
| Multiply | 0 | 0 | 4 | 0 |
| Floating point | 0 | 0 | 20 | 0 |

A performance improvement of 46% is seen using the proposed model. This is the motivation of this paper.

## 3. PROPOSED MODEL

Let there be n instruction types numbered from 1, 2... n. Let there be m processors numbered as $P_1,...,P_m$ . Consider the instructions that can be run simultaneously. Let a(i, j) be the number of functional units for instruction type j on processor $P_i$. The matrix A is of dimension mxn. Let t(i, j, k) be the time taken to execute one instruction of type j on functional unit k in processor $P_i$ . Let $x_{ijk}$ number of instructions of type j on functional unit k where k ranges from one to a(i, j) on processor $P_i$ . Let $y_{ij}$ be the number of instructions of type j scheduled on processor $P_i$. For any instruction of type j let $Q_j$ be the number instructions in the parallel code. Then we have

$$y_{1j} + y_{2j} + ... + y_{mj} \approx Q_j \quad (1)$$

i.e.

$$y_{11} + y_{21} + y_{31} + ... + y_{m1} \approx Q_1$$
$$y_{12} + y_{22} + y_{32} + ... + y_{m2} \approx Q_2$$
$$y_{13} + y_{23} + y_{33} + ... + y_{m3} \approx Q_3$$
$$....$$
$$y_{1n} + y_{2n} + y_{3n} + ... + y_{mn} \approx Q_n$$

From the proposed model we have for given i, j

$$x_{ij1} + x_{ij2} + ... + x_{ija(i,j)} \approx y_{ij} \quad (2)$$

We have

$$y_{11} \geq 0$$
$$y_{21} \geq 0$$
$$...$$
$$y_{m1} \geq 0$$
$$y_{12} \geq 0$$
$$y_{22} \geq 0$$
$$...$$
$$y_{m2} \geq 0$$
$$...$$
$$y_{1n} \geq 0$$
$$y_{2n} \geq 0$$
$$...$$
$$y_{mn} \geq 0 \quad (3)$$

Let $v_i$ be the maximum time to execute the instructions assigned to processor $P_i$. The objective is given by min(total time of execution of the parallel code)

The total time is given by the expression

Total time = max $(v_0, v_1, v_2, ..., v_{m-1})$

The calculation of $v_i$ is as follows.

$v_i$ =maximum time for computation on $P_i$ = maximum time for executing instructions of type 1, 2, .., n =max(maximum time taken on a(i,1) functional units, maximum time taken on a(i,2) functional units, maximum time taken on a (i,3), …, maximum time taken on a(i, n))

The objective function can be written as

min(max((max( $x_{111}$ t(1,1,1), $x_{112}$ t(1,1,2), $x_{113}$ t(1,1,3),…,

$x_{11a(1,1)}$ t(1, 1, a(1,1))), max( $x_{121}$ t(1,2,1), $x_{122}$ t(1, 2, 2),

$x_{123}$ t(1,2,3),…, $x_{12a(1,2)}$ t(1,2,a(1,2))),max( $x_{131}$ t(1,3,1),

$x_{132}$ t(1,3,2), $x_{133}$ t(1,3,3),…, $x_{13a(1,3)}$ t(1,3,a(1,3))),

…….

max( $x_{1n1}$ t(1,n,1), $x_{1n2}$ t(1,n,2), $x_{1n3}$ t(1,n,3),….,

$x_{1na(1,n)}$ t(1,n,a(1,n)))),

……..

( max( $x_{i11}$ t(i,1,1), $x_{i12}$ t(i,1,2), $x_{i13}$ t(i,1,3),…,

$x_{i1a(i,1)}$ t(i, 1, a(i,1))),

max( $x_{i21}$ t(i,2,1), $x_{i22}$ t(i, 2, 2), $x_{i23}$ t(i,2,3),…,

$x_{i2a(i,2)}$ t(i,2,a(i,2))),

max( $x_{i31}$ t(i,3,1), $x_{i32}$ t(i,3,2), $x_{i33}$ t(i,3,3),…,

$x_{i3a(i,3)}$ t(i,3,a(i,3))),

…….

max( $x_{in1}$ t(i,n,1), $x_{in2}$ t(i,n,2), $x_{in3}$ t(i,n,3),….,

$x_{ina(i,n)}$ t(i,n,a(i,n)))),

…….

( max( $x_{m11}$ t(m,1,1), $x_{m12}$ t(m,1,2), $x_{m13}$ t(m,1,3),…,

$x_{m1a(i,1)}$ t(m, 1, a(m,1))),

max( $x_{m21}$ t(m,2,1), $x_{m22}$ t(m, 2, 2), $x_{m23}$ t(m,2,3),…,

$x_{m2a(i,2)}$ t(m,2,a(m,2))),

max( $x_{m31}$ t(m,3,1), $x_{m32}$ t(m,3,2), $x_{m33}$ t(m,3,3),…,

$x_{m3a(i,3)}$ t(m,3,a(m,3))),

…….

max( $x_{mn1}$ t(m,n,1), $x_{mn2}$ t(m,n,2), $x_{mn3}$ t(m,n,3),….,

$x_{mna(i,n)}$ t(m,n,a(m,n))))))

subject to

$$t(1,1,1) \geq 0$$
$$t(1,1,2) \geq 0$$
$$\ldots$$
$$t(1,1,a(1,1)) \geq 0$$
$$t(1,2,1) \geq 0$$
$$t(1,2,2) \geq 0$$
$$\ldots$$
$$t(1,2,a(1,2)) \geq 0$$
$$\ldots$$
$$t(m,1,1) \geq 0$$
$$t(m,2,2) \geq 0$$
$$\ldots$$
$$t(m,n,a(m,n)) \geq 0$$
$$a(1,1) \geq 0$$
$$a(1,2) \geq 0$$
$$\ldots$$
$$a(1,n) \geq 0$$
$$\ldots$$
$$a(m,1) \geq 0$$
$$a(m,2) \geq 0$$
$$\ldots$$
$$a(m,n) \geq 0$$

$$x_{111} \geq 0$$
$$x_{112} \geq 0$$
$$x_{113} \geq 0$$
$$\ldots$$
$$x_{11a(1,1)} \geq 0$$
$$\ldots$$
$$x_{mn1} \geq 0$$
$$x_{mn2} \geq 0 \qquad (4)$$
$$\ldots$$
$$x_{mna(m,n)} \geq 0$$

Solving (4) for integer solutions gives the task allocation. This allocation ensures that the speed of the processors is utilized to the maximum extent.

## 4. SIMULATIONS

The proposed model allocates the tasks in the parallel segment of a program. The allocation is done by solving the optimization function given in (4). The optimization function can be solved using MS-Excel Solver package. The simulation of the proposed model does not require a system simulation as the proposed model is for task allocation. Consider the following system. There are four processors. There are four types of instructions namely load, store, multiply and floating point operation. The number of functional units for these processors for the four types of instructions is given by matrix A.

$$A = \begin{pmatrix} 2 & 1 & 3 & 4 \\ 3 & 2 & 1 & 5 \\ 1 & 3 & 0 & 2 \\ 1 & 2 & 2 & 1 \end{pmatrix}$$

Where a(i,j) denotes the number of functional units for instruction type j on processor $P_i$. The instructions are Load, Store, Multiply and Floating point operation as the columns of A. Thus there are five functional units of floating point type on processor 2. There are nine load instructions, six store instructions, seven multiply instructions, twenty floating point instructions to be scheduled in parallel.

The average time of execution in cycles on the functional units is given by the following.

Processor #1: $P_1$
Load unit 1: 3
Load unit 2: 2
Store unit 1: 2
Multiply unit 1: 6
Multiply unit 2: 5
Multiply unit 3: 4
Floating point unit 1: 20
Floating point unit 2: 20
Floating point unit 3: 12
Floating point unit 4: 10

Processor #2: $P_2$
Load unit 1: 3
Load unit 2: 2
Load unit 3: 4
Store unit 1: 1
Store unit 2: 2
Multiply unit 1: 5
Floating point unit 1: 15
Floating point unit 2: 20
Floating point unit 3: 14
Floating point unit 4: 10
Floating point unit 5: 10

Processor #3: $P_3$
Load unit 1: 2
Store unit 1: 3
Store unit 2: 2
Store unit 3: 3
Floating point unit 1: 10
Floating point unit 2: 9

Processor #4: $P_4$
Load unit 1: 2
Store unit 1: 3
Store unit 2: 3
Multiply unit 1: 6
Multiply unit 2: 8
Floating point unit 1: 12

The optimization function for the various instructions is given next. The notation used in Section 3 is used. For instruction type 1, the optimization function is

$$\min\begin{pmatrix} \max(x_{111}*3, x_{112}*2), \max\begin{pmatrix} x_{211}*3, \\ x_{212}*2, \\ x_{213}*4 \end{pmatrix}, \\ \max(x_{311}*2), \max(x_{411}*2) \end{pmatrix}$$

Subject to

$$x_{111} + x_{112} + x_{211} + x_{212} + x_{213} + x_{311} + x_{411} = 9 \quad (5)$$

Solving (5) for integer solutions, the allocation is 2,2 on $P_1$, 1,1,1 on $P_2$, 1 each on $P_3$ and $P_4$ giving a value of 6 cycles. For instruction type 2 the optimization function is given by

$$\min\begin{pmatrix} \max(x_{121}*2), \max(x_{221}*1, x_{222}*2), \\ \max(x_{321}*3, x_{322}*2, x_{323}*3), \\ \max(x_{421}*3, x_{422}*3) \end{pmatrix}$$

Subject to

$$x_{121} + x_{221} + x_{222} + x_{321} + x_{322} + x_{323} + x_{421} + x_{422} = 6 \quad (6)$$

Solving (6) for integer solutions gives 2 for $P_1$, 1, 2 for $P_2$, 3, 2, 3 for $P_3$ and 3 each for $P_4$ with a value of 4 cycles. For instruction type-3 the optimization function is

$$\min\begin{pmatrix} \max(x_{131}*6, x_{132}*5, x_{133}*4), \\ \max(x_{231}*5), \max(x_{431}*6, x_{432}*8) \end{pmatrix}$$

Subject to

$$x_{131} + x_{132} + x_{133} + x_{231} + x_{431} + x_{432} = 7 \quad (7)$$

Solving (7) gives 1, 1, 2 instructions on $P_1$, one on $P_2$ and one each on each functional unit of $P_4$ with a value of maximum of 8 cycles. For instruction type-4 the optimization function is

$$\min\begin{pmatrix} \max(x_{141}*15, x_{142}*20, x_{143}*12, x_{144}*10), \\ \max\begin{pmatrix} x_{241}*15, \\ x_{242}*20, x_{243}*14, \\ x_{244}*10, x_{245}*10 \end{pmatrix}, \\ \max(x_{341}*10, x_{342}*9), \max(x_{441}*12) \end{pmatrix}$$

Subject to

$$x_{141} + x_{142} + x_{143} + x_{144} + x_{241} + x_{242} + x_{243} + x_{244} + x_{245} + x_{341} + x_{441} = 20 \quad (8)$$

Solving (8) gives 1,1,2,2 on $P_1$, 1,1,1,2,2 on $P_2$, 2, 3 on $P_3$ and 2 on $P_4$ with a value of 27 cycles. The solution to the problem is hence max (4, 3, 8, 27) which is 27cycles.

## 5. CONCLUSION

An algorithm to allocate tasks in parallel segment of a program among heterogeneous CMP is proposed in this paper. The algorithm allocates based on the speed of computation on each processor. An optimization function is developed to allocate. The solution to the optimization function gives the allocation.

## 6. REFERENCES

[1] Ioannis Chatzigiannakis, Georgios Giannoulis, Paul Spirakis, Scheduling Tasks with Dependencies on Asymmetric Multiprocessors, PODC, '08.

[2] Nagesh B. Lakshminarayana, Jaekyu Lee, Hyesoon Kim, Age based scheduling for asymmetric multiprocessors, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.

[3] Qiong Cai , José González , Ryan Rakvic , Grigorios Magklis , Pedro Chaparro , Antonio González, Meeting points: using thread criticality to adapt multicore hardware to parallel regions, Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, 2008

[4] Saisanthosh Balakrishnan , Ravi Rajwar , Mike Upton , Konrad Lai, The Impact of Performance Asymmetry in Emerging Multicore Architectures, Proceedings of the 32nd annual international symposium on Computer Architecture, 2005, pp. 506-517

[5] S. Subha: An Algorithm for Parallel Execution of Loops in Chip Multiprocessor Caches, *ARTCom* 2009: 85-89

[6] S. Subha: A Scheduling Algorithm for Network on Chip, Advances in Computing, Control, and Telecommunication Technologies, International Conference on, pp.289-291