

Homework # 3

Due: Apr 3

Grain Size, Balance, and Cost-Performance in Multicore  
Processors

A big open research question in multicore design is how to choose the grain size of a multicore processor. Given a fixed chip area, should we build coarse-grained multicores, which have a few large cores, or fine-grained multicores, which have many smaller cores. In this homework you will explore this question.

When completing this homework and those in the future, keep in mind that Computer Architecture is part art and part science. Frequently, there are a number of ways to do the problems. As long as you state your assumptions clearly and your answer makes sense you will probably get full credit. *It is much better to think about a problem in depth and come to a general conclusion about the point of the problem than to spend a lot of time worrying about constant factors and boundary conditions.*

**Ex 1: Balance in Multicores**

A multicore processor is said to be *balanced* for a given algorithm with a problem of a given size if its processing, communication, and memory resources are each utilized to their fullest, without suffering any idle time (or unused space in the memory). (Notice this definition is slightly different from the one used by Kung.)

The notion of a balanced architecture allows us to assess the efficiency of an architecture. In other words, if a resource  $r$  is not utilized to its fullest, then the architect is better off apportioning some of the money (and design effort) spent on  $r$  to some other resource, thereby achieving additional performance.

Let us characterize a core using three parameters:  $(p, c, m)$ , where  $p$  is the processing power of the core in terms of the number of operations the processor can accomplish in a second,  $c$  is the number of *words* of data it can accept or send per second, and  $m$  is the number of words of memory per core (see Figure 1).

Note that, for now, we are ignoring physical constraints on how the words are communicated with other cores; just that, in any second, a total of  $c$  words can be communicated (sent or received) by a given core. Furthermore, we assume that the memory system of the core is designed with enough bandwidth such that each “operation” by the processor can read up to two operands from memory and write one operand back into memory, and that the communication system can add or remove words from memory at the peak transfer rate  $c$ . Practically speaking, clever design methods can achieve the above bandwidth re-

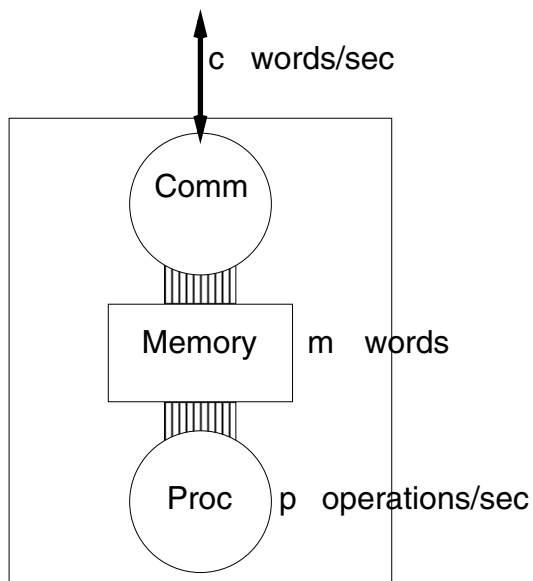


Figure 1: Characterization of a core in terms of its communication, processing, and memory.

quirements on the memory system by building memory hierarchies consisting of registers and caches for the processor, and buffers for the communication channels. Alternatively, if a processor needs to load two registers from memory and store a register's contents to memory to accomplish the above "operation," then each operation can actually take four processor cycles.

**Balance** A core is balanced for a given algorithm and a given problem size if the processing, communication, and memory requirements of the algorithm are exactly matched by the corresponding core parameters  $(p, c, m)$ .

As you will see in the ensuing exercises, the amount of memory required per core to achieve balance is related to the processing speed  $p$  and the communication speed  $c$ .

In the following exercises assume that computations can be overlapped completely with communication. In other words, if  $T_p$  is the time required to accomplish a given set of computations  $(R_p/p)$ , and  $T_c$  is the time required to satisfy the corresponding communication needs  $(R_c/c)$ , assume that the time for completion is

$$\max(T_c, T_p)$$

Note that if we did not allow overlap of communication and computation, as in a ensuing exercise, then the time for completion would be  $(T_c + T_p)$ . As stated earlier, we are assuming that the time for memory accessing is folded into the time for computation  $T_p$ .

(1) Suppose that for a given algorithm with a given problem size, the memory required per processing core is  $R_m$ . Identify the relationships between  $T_c$ ,  $T_p$ ,  $R_m$ , and  $m$  for

architectural balance for that algorithm. (Since the rest of the exercises are based on these relationships, the answers to this question are given at the end of this homework!)

(2) If the problem size for a Jacobi relaxation with the basic iteration step

$$A_{i,j} = \frac{A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1}}{4}$$

is  $N$  (that is, if we have an  $\sqrt{N} \times \sqrt{N}$  grid), and the problem is subdivided among  $P$  processing cores in square blocks, what is the memory requirement  $R_m$  per core? What is the processing requirement  $R_p$  per core? (Assume that computing a new  $A_{i,j}$  takes one cycle given all the values in the above equation are already present in memory.) How many words of data  $R_c$  need to be input (or output) from each core? Focus on a single iteration of Jacobi, and ignore the space required to store values fetched from other cores during the computation.

(3) For a single relaxation step of the Jacobi relaxation algorithm, given  $N$  and  $P$ , determine the relationship between  $c$  and  $p$  for architectural balance.

(4) Consider a processing core with  $p = 2 \times 10^9$  ops/sec,  $m = 8K$  words,  $c = 10^8$  words/sec, and  $P = 100$ . For what problem size  $N$  are all core resources (approximately) fully utilized? Alternatively, show that balance cannot be achieved with any problem size.

(5) Now, consider a balanced core whose processing power  $p$  is increased by a factor  $\alpha$  over the value for the application above. If  $c$  must remain unchanged, what are all the possible ways to re-balance the core? (Depending on the constant factors you use, you may or may not be able to achieve complete balance. Explain your methodology.)

(6) On the other hand, if  $c$  were increased by  $\alpha$ , by what factor should the processor parameter  $p$  be changed to regain balance?

**Size Efficiency** A core that requires a larger problem size to achieve balance will have a relatively poorer size efficiency than a core that can achieve balance with a smaller problem size. Put another way, a core that requires more memory to achieve balance for the same processing power has a poorer size efficiency than a core that needs a smaller amount of memory for the same processing power.

(7) For Jacobi, show that the relative size efficiency of machines can be deduced from the ratio  $p/c$ . Which of the two cores in problems (5) and (6) would you classify as being more size efficient for the Jacobi problem?

### Ex 2: More on Balance

Consider the following machine characterized by the three parameters

$(p, c, m)$ :  $(6 \times 10^9 \text{ ops/sec}, 0.5 \times 10^9 \text{ words/sec}, 256K \text{ words})$

Also consider an  $N$  point FFT algorithm. Assume that there are  $P$  processors and that the computations are evenly apportioned to each processing core using the decomposition

suggested in the paper by Kung handed out in class. (Alternatively, you may also choose to use your own partitions to compute communication and computation requirements. Clearly describe your choice, and do not be worried if you are off by a constant factor.) Assume as before all computations can be overlapped with communications.

- (1) For the FFT algorithm, indicate whether there is *any* problem size that allows the machine to be balanced.
- (2) Suggest how to balance the machine in the direction of making the machine more size efficient, by indicating which values you would change and why.
- (3) Suggest how to balance the machine in the direction of making the machine less size efficient, by indicating which values you would change and why.

### **Ex 3: Choosing the Grain Size to Optimize Cost-Performance**

As defined in the Simplefit paper, suppose that the cost of processing performance can be represented as  $K_p(p)$ , the cost of communication can be represented as  $K_c(c)$ , and the cost of memory as  $K_m(m)$ . Then the cost  $K$  of a machine with  $P$  cores, each with parameters  $(p, c, m)$ , can be written as

$$K = P(K_c(c) + K_p(p) + K_m(m))$$

As defined in class, suppose that

- $K_p(p) = K_{ps} \times p_0$ , a constant,
- $K_c(c) = K_{cs} \times c$ , proportional to  $c$ , and
- $K_m(m) = K_{ms} \times m$ , also proportional to  $m$ .

The parameters  $K_{ps}, K_{cs}, K_{ms}, p_0$  are known constants.

- (1) Suppose we have a budget of  $B$  and a Jacobi problem of size  $N$ , write a constraint relation between the core parameters and the cost  $B$ . Write an expression for the running time,  $T$ , assuming computations and communications are completely overlapped.
- (2) Find the optimal number of processors  $P$  and the core parameters that will minimize the running time, of course, subject to the cost constraint. Note that the processor speed is fixed at  $p = p_0$ .

You should work out the problem algebraically to the point where you have  $n$  equations and  $n$  unknowns. You are not required to simplify the resulting equations. If you are ambitious, you may show the solution method using a graph to depict the constraint and costs. (Hint: for algebraic solutions, use the Lagrange multiplier method outlined in class to solve optimization problems with a constraint).

- (3) The processor model in the above problem assumed fixed cost, fixed speed processors. It further assumed that the cost of communications and the cost of memory was

proportional to the communication bandwidth and the size of memory respectively. Are the above models for processors, communications, and memory reasonable? Briefly justify your answer.

#### **Ex 4: Non-Overlapped Computations and Communications**

When communications and computations are not overlapped, the running time can be expressed as the sum  $T_p + T_c$ . As you might have realized, it's hard to apply the notion of balance in such a situation.

For the cost constraint that you used in the previous problem, find the optimal ratio of processing power  $p$  to communication bandwidth  $c$ , to minimize the running time.

Again, simply show the  $n$  equations in  $n$  unknowns that must be solved. You are not required to simplify the equations.

#### **Ex 5: SIMD versus MIMD Computational Models**

Consider the following `do_in_parallel()` construct<sup>1</sup>, where  $A$  is a two dimensional  $N \times N$  array, with each element distributed (one word to each core) in a 2-D array of processing cores, also of size  $N \times N$ . Let the cores in the processor array be denoted  $(i, j)$ , where a core's PID is  $Ni + j$ . Let the elements in the data array be denoted  $A[x, y]$ . Then the distribution is such that processor  $(i, j)$  gets element  $A[i, j]$ .

---

<sup>1</sup>The `do_in_parallel(func, [ args ... ])` function creates one process on each processor and executes the processes concurrently. The process can access its process ID by calling `my_pid()`. Each process terminates after it executes the function named in the `do_in_parallel()`. The `shared` attribute indicates that a variable is to be placed in a shared address space and accessible by any process.

```

shared int A[N][N];

void
compute(void)
{
    i = my_pid() / N;
    j = my_pid() % N;
    ...
    A[a0*i + b0*j + c0][a1*i + b1*j + c1] = 2;
    ...
}

...
do_in_parallel(compute);

```

Assume the processor array's network operates in a purely SIMD<sup>2</sup> fashion. That is, in any given network cycle, every processing core must issue an address *relative* to its location to access its array element. If different relative addresses are issued, the SIMD controller must issue multiple network cycles to complete the instruction; for example, executing 'A[i^8][j] = 2' would take two network cycles.

(1) Derive conditions on the relative values of  $a_0$ ,  $a_1$ ,  $b_0$ ,  $b_1$ ,  $c_0$ , and  $c_1$ , such that all the array references made in the above parallel 2-D loop can be accomplished concurrently, in one network cycle.

**Hint:** the relative address used by processor  $(i, j)$  to pick its array element in the above example is:

$$[a_0 * i - i + b_0 * j + c_0, a_1 * i + b_1 * j - j + c_1]$$

**Hint:** the values of  $a_0, b_0, \dots$  that work may actually be very restricted and simple; essentially, they are constraints on what can be done with a SIMD model.

(2) In the worst case, how many cycles will it take to accomplish all the array references?

Answer to question (1) in Exercise 3:  $T_c = T_p$  and  $R_m = m$ . For full utilization the completion time must equal the individual completion times for communication and computation. Furthermore, the memory of the core must be fully utilized.

---

<sup>2</sup>Note that we distinguish between a SIMD architecture and a SIMD instruction set. A SIMD architecture is a parallel computer architecture distinguished by a network of parallel processors each connected to a shared data memory and single instruction sequencer. In each clock cycle the instruction sequencer issues the same instruction to every processor. A SIMD instruction set, however, allows programmers to specify that a single instruction is to be performed on multiple sub-word data values in parallel. This homework question refers to the SIMD architecture and not the instruction set.