ECE669: Parallel Computer Architecture
Fall 2004 **Handout #3**

Homework # 3 Due: October 20

Computational Models, Balance, Cost-Effectiveness

Ex 1: Physical Transparency

As defined in class, *physical transparency* is the property of an architecture that allows machines of different sizes to be built simply by replicating the same basic node. Physical transparency also requires that if an application with some problem size runs on a machine with a given number of nodes, then the same application with the same problem size must also run on a machine with the same architecture, but with any other number of nodes. Bear in mind that physical transparency does not force any sort of performance guarantee.

Suppose that a N point Jacobi relaxation runs on a 1000-processor machine. If you are designing a machine to support physical transparency, what issues must you consider. Consider, for example, supporting the ability to run the same application with the same problem size on the machine with a single processing node. You may assume that you have a disk large enough to hold the entire problem.

Ex 2: Balance in Parallel Computer Architecture

A parallel computer 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 is useful because it 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.

The notion of balance is useful in choosing the *grain size* of a node in a parallel computer. At the present time, a crisp definition of the grain size of a node does not exist, and architects are prone to using indiscriminately terms such as fine grain, medium grain, or coarse grain. There is, however, the vague notion that machines that require "lots" of memory per node are coarse grain, while those that require "little" memory per node are fine grain.

Let us take a stab at defining precisely the notion of grain size. First, let us characterize a node using three parameters: (c, m, p), where p is the processing power of the node 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 node (see Figure 1).

Note that, for now, we are ignoring physical constraints on how the words are com-

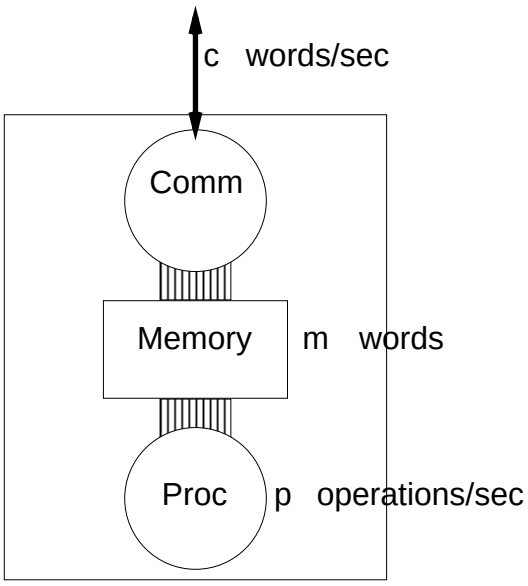

Figure 1: Characterization of a node in terms of its communication, processing, and memory.

municated with other nodes; just that, in any second, c words can be exchanged with "other" nodes. Furthermore, we assume that the memory system of the node 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 requirements 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.

As defined earlier, a node 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 node parameters (c, m, p).

Based on the above definition of balance, we can come up with a crisp notion of the relative grain sizes of various nodes. *Intuitively, a node that requires a larger problem size to achieve balance will have a relatively coarser grain than a node that can achieve balance with a smaller problem size.* Put another way, a node that requires more memory for the same processing power has a coarser grain than a node that needs a smaller amount of memory for the same processing power.

As you will see in the ensuing exercises, the amount of memory required per node to achieve balance is related to the ratio of processing speed p to communication speed c. Accordingly, I propose that a node be classified as fine-grain if p/c is about 0.1 or less (in

units of operations per word), medium-grain if p/c is about 1, and coarse-grain if p/c is about 5 or more.

Thus, for example, a machine using a 40 MHz processor capable of about 10 million operations per second, with about 10M words per second of communication bandwidth would be classified as a medium grain machine.

It is not an oversight that the amount of memory per node m has been left as a free variable in the determination of whether a machine is fine grain or coarse grain. The assumption is that, for any given algorithm, m is uniquely determined by the problem size required to achieve balance. This problem size is in turn related to the values of p and c. (Suggestions on other proposals for a definition of fine grain, medium grain and coarse grain are invited.)

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, and T_c is the time required to satisfy the corresponding communication needs, 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 node is S_m. Identify the relationships between T_c, T_p, S_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 nodes in square blocks, what is the memory requirement per processing node? What is the processing requirement? How many words of data need to be input or output from each node? Focus on a single iteration of Jacobi, and ignore the space required to store values fetched from other nodes 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 node with $p = 8 \times 10^6$ ops/sec, $m = 10^6$ words, $c = 4 \times 10^4$ words/sec, and $P = 10^4$. For what problem size N are all node resources fully utilized? Alternatively, show that balance cannot be achieved with any problem size.

(5) Now, consider a processing node whose processing power p is increased by a factor α over the value in the problem above. If c must remain unchanged, what are all the possible ways to rebalance the node?

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

(7) Which of the two nodes in problems (5) and (6) would you classify as having a finer grain size?

(8) Suppose we have a Jacobi problem of size N, what are the relative values of c, m, and p, that must be chosen as a function of the number of processing nodes P, so that the nodes remain balanced. Of course, assume $1 \leq P \leq N$.

Ex 3: Grain Size

Consider the following machines characterized by the three parameters, (c, m, p), specified below.[1]

A: $(40 \times 10^6, 2 \times 10^6, 10 \times 10^6)$

C: $(2.5 \times 10^6, 0.5 \times 10^6, 0.016 \times 10^6)$

D: $(10 \times 10^6, 4 \times 10^6, 10 \times 10^6)$

E: $(2 \times 10^6, 4 \times 10^6, 10 \times 10^6)$

J: $(16 \times 10^6, 0.25 \times 10^6, 3 \times 10^6)$

M1: $(40 \times 10^6, 0.016 \times 10^6, 3 \times 10^6)$

M2: $(50 \times 10^6, 5 \times 10^6, 10 \times 10^6)$

N1: $(80 \times 10^6, 0.01 \times 10^6, 10 \times 10^6)$

N2: $(160 \times 10^6, 8 \times 10^6, 10 \times 10^6)$

P: $(19.2 \times 10^3, 0.064 \times 10^6, 0.1 \times 10^6)$

Also consider these applications:

1. the ubiquitous Jacobi, with problem size N, i.e., with a $\sqrt{N} \times \sqrt{N}$ grid.
2. matrix multiply of two N element matrices, i.e, each is a $\sqrt{N} \times \sqrt{N}$ matrix. (For this focus on the problem of multiplying two sub matrices of the appropriate size.)

Also assume that there are P processors and that the computations are evenly apportioned to each processing node using the decompositions suggested in the paper by Kung handed out in class. Assume as before all computations can be overlapped with communications.

(1) Pick any two of your favorite machines from those given above. For each of the applications, indicate whether there is *any* problem size that allows the machines to be balanced. If not, define the degree of imbalance *for a given algorithm* as the ratio b_i/b, where

[1] Any similarity with existing machines is, well, intentional.

- b is the required ratio of processing to communication for balance when the problem size is chosen to fully utilize memory, and
- b_i is the ratio of processing to communication provided by the machine.

(2) Suggest how to balance the machines in the direction of making the machine more coarse, and

(3) in the direction of making the machines more fine.

Ex 4: SIMD versus MIMD Computational Styles

Consider the following SIMD application:

The encoding of MPEG video requires a "motion estimation" step, in which portions of two sequential frames (images) are compared to find the overall difference. With this information, the MPEG encoder can determine if that portion of the image has changed.

A simple description of this is shown in the following pseudocode,

```
for all i from 1 to x
{
  for all j from 1 to y
  {
    sum = ( A[i,j] - B[i,j] ) + sum ;
  }
}
```

where A and B are the same portion of two subsequent frames. Each element of A and B is a pixel, and the portion being compared is (x, y) pixels in size.

This is a job for SIMD. Say our MPEG encoder has an array of processing elements (PE) which each contain one ALU. There are $x \times y$ PEs.

(1) Since each PE contains only one ALU, how many instructions will it take to compute the sum in the above pseudocode?

(2) How should A and B be distributed to the PEs so that they may all perform the subtraction operation in one cycle?

Since there is only one running sum, the PEs would have to access that memory location one at a time, which would take $x \times y$ cycles. In the real MPEG encoder, this problem is solved in a clever way by accumulating the sum while new data is being shifted into the PE array. This requires a certain amount of communication between the PEs. The PEs use a *relative* address to access data outside of its own location.

(3) Describe, in your own words or with pseudocode, how a PE might be programmed to use data from its neighbor to the North (above).

Answer to question (1) in the exercise on architectural balance: $T_c = T_p$ and $S_m = m$. For full utilization the completion time must equal the individual completion times for communication and computation. Furthermore, the memory of the node must be fully utilized.

Ex 5: Traveling Salesman Use Little's Algorithm to locate an initial tour for the six-node undirected graph shown on page 14 of the Lecture 10 slides. When reducing the graph, first reduce rows and then columns. What is the total cost of the path?

Ex 6: Culler problem 3.6

Ex 7: Culler problem 3.14, part a