

Homework #3 Solutions

Ex 1: Balance in Multicores

(1) By our definition of architectural balance, we don't want any wasted resources. Since T_c can be completely overlapped with T_p , we want $T_c = T_p$. Also, we want just enough memory for the problem on each node, so $m = R_m$.

(2) The memory requirement is $R_m = \frac{N}{P}$. Note this indicates in-place computation.

The processing requirement is $R_p = \frac{N}{P}$, assuming one unit of computation for each $A_{i,j}$ calculation. Constant factors indicating operations per iteration don't affect the results very much.

The communication requirement is $R_c = 4\sqrt{\frac{N}{P}}$ if we count only reads, or twice that if we count reads and writes. This follows from homework #1.

(3)

$$\frac{p}{c} = \frac{R_p}{R_c} = \frac{\frac{N}{P}}{4\sqrt{\frac{N}{P}}} = \frac{1}{4}\sqrt{\frac{N}{P}}$$

(4) Keeping in mind that for this processing node, $p = 2 \times 10^9$ ops/sec, $m = 2^{13}$ words, $c = 10^8$ words/sec, and $P = 100$ it is possible to solve for N using the relation derived above.

$$\frac{2 \times 10^9}{10^8} = \frac{1}{4}\sqrt{\frac{N}{P}}$$

$$N = \left(\frac{2 \times 10^9}{10^8} \times 40 \right)^2 = 640000$$

$$R_m = \frac{640000}{100} = 6400$$

This value of R_m is close to the given memory size of $m = 8K$ words.

(5) By looking at the p - c relation, one can find ways to rebalance the architecture when p is increased by a factor α

$$\frac{p}{c} = \frac{1}{4} \sqrt{\frac{N}{P}}.$$

If c remains unchanged, the architecture can be rebalanced by increasing the problem size by a factor of α^2 , decreasing P by a factor of α^2 , increasing N by α and decreasing P by α , or decreasing N by α and increasing P by α . Of course, along with these changes, the amount of memory m needs to be adjusted such that $m = \frac{N}{P}$ is still true. All of these choices lead to an increase in memory by a factor of α^2 .

(6) Assuming that the architecture was previously balanced, if c were increased by a factor of α , then p needs to be increased by a factor of α as well if N , P , or m are held constant.

(7) Since a relative judgement on size efficiency can be made by comparing the amount of memory each node contains, the more size-efficient node will have less memory. Since rebalancing in (5) requires a memory adjustment, namely an increase in memory by a factor of α^2 , a node in (6) may be classified as being more size-efficient than a node in (5).

Another way to see this is to use the p/c ratio metric. Rebalancing in (5) increased the p/c ratio by a factor of α while rebalancing in (6) left the p/c ratio unchanged. Since (5) has a greater p/c ratio, the node in (5) can be considered to be less size-efficient than the node in (6).

Ex 2: More on Balance

(1) Machine A is analyzed below. Analysis for other machines follows in a similar fashion. For architectural balance, $m = R_m$ and $T_c = T_p$.

The basic premise of the FFT algorithm is to break up an N -point computation into a number of 2-point computations. For this problem, we want to break up the N -point computation into a number of M -point computations, where M is the size of the local memory of a node. Assuming that N and M are powers of 2 (you can zero-pad for this to be true), then the N -point computation can be partitioned among P nodes by assigned $M = \frac{N}{P}$ points to each node.

On each iteration, each node fetches M values from other nodes and computes a new M -point FFT. The computation requires M amount of memory, ignoring storage for fetched values, since it executes in place. The communication requirement will be $2M$ or on the order of M to account for all the fetches and sends. The processing requirement for each node is a little more complicated since each node computes an M -point FFT. The M -point computation can be divided into a number of 2-point computations. At each level of the computation, M operations are performed. The number of levels depends on how many times M can be divided by 2 or $\log_2 M$. This means the processing requirement for a node is $M \log_2 M$. These values all agree with Kung's estimates. Then, the requirements for a balanced architecture for an FFT algorithm are:

$$m = M$$

$$\begin{aligned}
&= \frac{N}{P} \\
\frac{p}{c} &= \log_2 M \\
&= \log_2 \frac{N}{P}
\end{aligned}$$

(p,c,m): $(6 \times 10^9, 0.5 \times 10^9, 256K)$

$$\begin{aligned}
\log_2 \frac{N}{P} &= \frac{p}{c} \\
&= \frac{6 \times 10^9}{.5 \times 10^9} \\
&= 12 \\
N &= 12P.
\end{aligned}$$

Now check if memory size agrees with this value of N .

$$\begin{aligned}
m &= \frac{N}{P} \\
m &= \frac{2^{12}P}{P} \\
&= 2^{12} \\
&= 4K \\
&\neq 256K.
\end{aligned}$$

So, there is no problem size which will balance this machine for the FFT algorithm.

(2) To balance the machine in the direction of making it more size efficient, we can simply decrease the memory size. If $m = 2$, for example, the machine would be very size-efficient. We could also increase p/c somewhat at the same time so as to meet at a moderate value of m .

(3) To balance the machine in the direction of making it less size efficient, we simply increase p/c .

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

(1) The constraint equation is:

$$B = P(K_{ps}p_0 + K_{cs}c + K_{ms}m)$$

For a Jacobi problem of size N , the running time T will be the maximum of the processing time and communication time, *i.e.*,

$$T = \max \left(\frac{N}{pP}, \frac{1}{c} \sqrt{\frac{N}{P}} \right)$$

We also need the memory to hold the problem size:

$$m = N/P$$

(2) Since the processing speed is fixed, the running time will be minimized if we constrain the communication time to same as the processing time, *i.e.*,

$$\min(T) = \frac{N}{pP} = \frac{1}{c} \sqrt{\frac{N}{P}}$$

Any additional money spent on increasing c to decrease communication time will be wasted; essentially you want to spend money on increasing c up to this point and then throw the resources into some other parameter. We can incorporate this requirement as another constraint, $\frac{N}{pP} = \frac{1}{c} \sqrt{\frac{N}{P}}$, on the minimization of T .

Similarly, we constrain the memory to be equal to the problem size. Any additional money spent on memory beyond the size of the problem is wasted. We incorporate this as the constraint $m = N/p$.

Using the Lagrange multiplier method,

$$\begin{aligned} T &= \frac{N}{p_0P} + \lambda_1(B - PK_{ps}p_0 - PK_{cs}c - PK_{ms}m) + \lambda_2 \left(\frac{N}{p_0P} - \frac{1}{c} \sqrt{\frac{N}{P}} \right) + \lambda_3 \left(m - \frac{N}{P} \right) \\ \frac{\partial T}{\partial P} &= -\frac{N}{p_0P^2} - \lambda_1(K_{ps}p_0 + K_{cs}c + K_{ms}m) - \lambda_2 \left(\frac{N}{p_0P^2} - \frac{1}{2cP} \sqrt{\frac{N}{P}} \right) - \lambda_3 \frac{N}{P^2} \\ \frac{\partial T}{\partial c} &= -\lambda_1 PK_{cs} + \lambda_2 \frac{1}{c^2} \sqrt{\frac{N}{P}} \\ \frac{\partial T}{\partial m} &= -\lambda_1 PK_{ms} + \lambda_3 \\ \frac{\partial T}{\partial \lambda_1} &= B - PK_{ps}p_0 - PK_{cs}c - PK_{ms}m \\ \frac{\partial T}{\partial \lambda_2} &= \frac{N}{p_0P} - \frac{1}{c} \sqrt{\frac{N}{P}} \\ \frac{\partial T}{\partial \lambda_3} &= m - \frac{N}{P} \end{aligned}$$

Setting each of the partial derivatives of T to zero gives you 5 equations with 5 unknowns ($P, c, m, \lambda_1, \lambda_2$) which can easily be solved using standard linear system methods.

(3) The above cost model is not very reasonable. First of all, fixed cost, fixed speed processors are unrealistic. In general, faster processors cost more money and folding this fact into the cost model will make the optimization more flexible. Communication cost is more exponentially related to communication speed, rather than linearly related. This is because communication speed is closely related to network bandwidth and bandwidth is exponentially related to cost due to increased die sizes, pin counts, etc. It also can be said

that the amount of memory is not linearly related to cost. Memory cost can be perceived as cost per bit. This metric tends to be significantly smaller for higher, density memory. However, memory access speed may be considered to be linearly related to cost, so that memory cost should also be directly related to p (maybe linearly related) as well.

Ex 4: Non-Overlapped Computations and Communications

$$B = P(K_{ps}p_0 + K_{cs}c + K_{ms}m)$$

$$T = \frac{N}{pP} + \frac{1}{c} \sqrt{\frac{N}{P}}$$

$$m = \frac{N}{P}$$

$$T = \frac{N}{p_0P} + \frac{1}{c} \sqrt{\frac{N}{P}} + \lambda_1(B - PK_{ps}p_0 - PK_{cs}c - PK_{ms}m) + \lambda_2 \left(m - \frac{N}{P} \right)$$

$$\frac{\partial T}{\partial P} = -\frac{N}{p_0P^2} - \frac{1}{2cP} \sqrt{\frac{N}{P}} - \lambda_1(K_{ps}p_0 - K_{cs}c - K_{ms}m) - \lambda_2 \frac{N}{P^2}$$

$$\frac{\partial T}{\partial c} = -\frac{1}{c^2} \sqrt{\frac{N}{P}} - \lambda_1 PK_{cs}$$

$$\frac{\partial T}{\partial m} = -\lambda_1 PK_{ms} + \lambda_2$$

$$\frac{\partial T}{\partial \lambda_1} = B - PK_{ps}p_0 - PK_{cs}c - PK_{ms}m$$

$$\frac{\partial T}{\partial \lambda_2} = m - \frac{N}{P}$$

Ex 5: SIMD versus MIMD Computational Models

(1) The relative address is:

$$[(a_0 - 1) * i + b_0 * j + c_0, a_1 * i + (b_1 - 1) * j + c_1]$$

In order for the above relative address to be the same across all processors (i, j) it is necessary to use the following parameter values:

$$a_0 = 1$$

$$b_0 = 0$$

$$a_1 = 0$$

$$b_1 = 1$$

$$c_0 = \text{constant}$$

$$c_1 = \text{constant}$$

(2) In the worst case, all processors could choose the same element to access. Thus, N^2 instruction cycles would be needed; one for each element access.