White paper draft. June, 1997.

Assigning PEs to Loosely Synchronous Computations at Runtime

David R. O'Hallaron

1 Overview

This note describes the runtime information that we will need from Remulac in order to assign PEs to loosely synchronous computations such as the sparse matrix vector product (SMVP).

A simple model characterizes the performance of PEs participating in an SMVP. The model relies on application and system properties, and it characterizes the local floating point performance as well as the communication performance across the network interface. The key system parameters in the model are: (1) average time per SMVP flop (T_f), (2) the overhead per communicated block (T_l), (3) the overhead per additional communicated word (T_w), and (4) the amount of available memory (M).

Our aim is to use the model to allocate PEs at runtime in order to satisfy certain performance objectives. These objectives might be expressed in relative or absolute terms. For example, we might want the program to spend at least 20% of its time computing (rather than communicating). Or we may require each SMVP to finish in under 1 second. In either case, a key requirement is getting accurate runtime estimates of the system properties T_f, T_l, T_w, and M.

Section 2 summarizes the performance model. Section 3 sketches an approach for using the model to allocate compute resources.

2 Performance model

Running time for the SMVP is given by

$$T_{mv} = T_{comp} + T_{comm} \leq FT_f + (B_{max}T_l + C_{max}T_b) \tag{1}$$

where T_{comp} is the time required for all processors to finish their computation phases and T_{comm} is the time required for all PEs to finish their communication phases. (For reference, the symbols in the writeup are summarized in Figure 1). Equation 1 makes a conservative simplifying assumption that the PE with the most communicated blocks is also the PE with the most communicated words. It can be shown that this assumption overestimates the running time of the communication phase by at most

$$\beta = 1 + \min_{i \text{ is a processor}} \left[\max \left\{ \frac{(B_{max} - B_i)C_{max}}{B_{max}C_i}, \frac{(C_{max} - C_i)B_{max}}{B_i C_{max}} \right\} \right] \tag{2}$$

where $1 \leq \beta \leq 2$. For the Quake applications, β is usually 1 and the largest value we have seen is 1.15.

The relationship between the average time for a floating point operation and the average time for a communicated word is given precisely by

$$T_c = \frac{F}{C_{max}} \left(\frac{1 - E}{E} \right) T_f. \tag{3}$$

The average time per communicated word can be approximated in terms of the application parameters and lower-level communication parameters in the system:

$$T_c \leq \frac{B_{max}}{C_{max}} T_l + T_w \tag{4}$$

T_{comm} is overestimated by at most a factor of β, and since $T_c = T_{comm}/C_{max}$, T_c is also overestimated by at most a factor β.

Symbol	Description
T_{SMVP}	running time for SMVP
T_{comp}	running time for SMVP computation phase
T_{comm}	running time for SMVP communication phase
F	flops per processing element (PE)
C_i	communicated words for PE i
C_{max}	max number of communicated words per PE
B_i	communicated blocks for PE i
B_{max}	max number of communicated blocks per PE
T_f	time per flop (inverse of sustained FLOPS)
T_c	time per communication word (inverse of sustained bandwidth)
T_l	time per block (block latency)
T_w	time per additional block word (inverse of block bandwidth)
M	amount of available memory
β	error bound on communication time
E	efficiency (T_{comp}/T_{mv})

Figure 1: Symbols.

3 Assigning PEs to the Quake application

Consider the following scenario. The application is dominated by the SMVP and is static in the sense that mesh is does not change over time and there is a fixed amount of work per node per timestep. However, the computing platform does change over time as PEs come online and offline, as PE's become busy and idle, and as available memory is allocated and deaxllocated. The problem is to automatically assign PEs to the Quake application, based on the current configuration of the computing platform, so as to achieve some performance goal.

Given a basic capability to determine T_f, T_l, T_w, and M at runtime, then this problem is fairly straightforward, as a couple of examples illustrate. For each example, assume that we have precompiled instances of the application for a range of partitions and that we have also precomputed the set of application properties F, B_{max}, C_{max} for each of these partitions.

Example 1. Suppose that the absolute performance goal is for each SMVP to run in under 1 second on the fewest number of PEs. Then we can plug runtime values of T_l and T_w into Equation 4 for different partitions to determine the appropriate set of PEs. Of course, we must also ensure that the SMVP will fit in the available memory of these processors, so we must also know the system parameter M.

Example 2. Suppose that the relative performance goal is to find a collection of PEs that will run the code at 80% efficiency. Then we can plug the runtime value of T_f into Equation 3 for different partitions to arrive at a partition with the appropriate efficiency.