University of Puerto Rico
Mayagüez Campus

Mathematics Department

Experimental Validation of Bulk Synchronous Parallel on Origin 2000

Elio Lozano
Advisor Dr. Edgar Acuña

March 9, 2003

Abstract

Experimental data validating some of the proposed parallel computation models on SGI Origin 2000 is presented. This Architecture is characterized by a large bandwidth and a relatively large startup cost of a message transmission, which makes it extremely important to employ bulk transfers. The models considered are the **BSP** model, in which it is assumed that all messages have a fixed short size; and the message passing model **MPI**, in which the message passing can be synchronous or asynchronous.

1 Introduction

Experimental validation of models of parallel computation is extremely valuable. To emphasize this, Snyder [Sny95] wrote "in the absence of evidence that it make accurate predictions, what serious interest can there be in either model, or algorithms developed with it?" Although some experimental result investigating the prediction capabilities of parallel model have been published, these results were often unsatisfactory for the following reasons. First, sometimes the experimental data was collected on only one platform. In other cases, the result were gathered on systems with only a few processors. Third, often the communication overhead was only a small part of the total execution time. Since communication cost is the only cost captured in detail by most models, this provides no support for their predictive capabilities. The Models considered in this paper are the Bulk Synchronous Parallel (BSP) and Message Passing Interface (MPI). In this section these models are briefly described. The BSP model [Val90] consist of p processors, a communication medium that can deliver message point to point, and a mechanism for barrier synchronization. BSP computations are organized in *supersteps*, separated by *barrier synchronization*. In a superstep, a processor can perform local operations, and send and receive some messages. Two parameters are used to model communication cost: (1) the latency/synchronization cost L, and (2) the computational-to-communication through radio g.

Central to the BSP model is the concept of an h-*relation*: a communication pattern in which each processor sends and receives at most h messages. The parameters g and L are such that an arbitrary h-relation followed by a barrier synchronization takes $g \cdot h + L$ time. The cost of a superstep is therefore $w + g \cdot h + L$, where w is the maximum amount of local work performed by any processors, and h is the maximum number of messages sent or received by any processor. In the BSP model [Val90], all message have a fixed short size (essentially the word size of the machine). However, it is well known that on more parallel architectures there is a large startup cost associated with a message transmission.

The Message Passing Interface [Pac97] is the most commonly used method for programming distributed-memory systems. We saw that message passing can be synchronous or asynchronous. If a

system provides *buffering*, then the system can copy the sender's message to a system buffer, and the sender can continue with its work. However, if there is no buffering, the processes must synchronize; i. e. the sender must receive permission from the receiver to transmit the message . Message-Passing functions can also be either blocking o no blocking. In blocking message passing, a call to a communication function won't return until the operation is complete. For example, a blocking receive function will not return until the message has been copied into the user process's memory. No blocking communication consists of two phases. During the first phase a function is called that starts the communication. During the second phase another function is called that complete the communication.

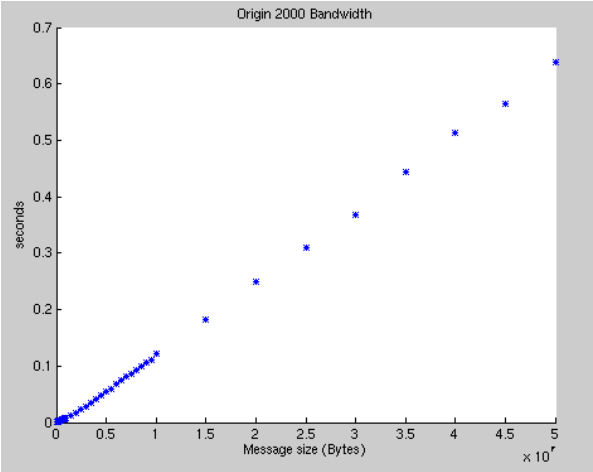

Figure 1: Ping pong benchmark for Origin 2000

2 Machine Description

For the experiments, we used the SGI Origin 2000 located at the high performance computing facility of the Puerto Rico University (Mayagüez Campus). It is an 8 processor cache coherent non-uniform memory access machine (ccNUMA)[Lau98]. Each processor is RISC R10000, with 1GB main memory, is a 64 bit chip running at 250 Mhz capable of 390 Mflops (2 flops per cycle) [Sai96]. Each processor has 64 floating point registers and 64 integer registers. The machine has separately level one instruction and data caches, but a unified level two cache. The level 1 cache is 32KB with two ways set associativity. The instruction cache has a line size of 64 bytes, while the data cache line size is smaller at 32 bytes. The 4 MB level two cache is two way associative with a 128 byte line size.

3 Experiments

We implemented several simple benchmark programs using the native message passing library. Fig 1 shows one of two experiments. In the first experiment, two processors repeatedly exchange a message between them (the well known "ping-pong" benchmark). The results of this experiment are labeled "send/receive". In the second experiment, both processors send and receive a message simultaneously (the "ping-pong double" benchmark). The results of the experiment are labeled "send + receive". It can be seen that the time taken by both experiments is well approximated by the measured data points, we use least square to fit a line to the resulting (message-size,time) pairs.

The intercept will approximate t_s and the slope t_c. The equation is given by:

$$time = t_s + t_c(message\_size)$$

where:

$$t_s = \frac{n(\sum_{i=1}^{n} n_i^2) - (\sum_{i=1}^{n} n_i)^2}{n(\sum_{i=1}^{n} n_i t_i) - (\sum_{i=1}^{n} n_i)(\sum_{i=1}^{n} t_i)}$$

$$t_c = \frac{(\sum_{i=1}^{n} n_i^2)(\sum_{i=1}^{n} t_i) - (\sum_{i=1}^{n} n_i t_i)(\sum_{i=1}^{n} n_i)}{n(\sum_{i=1}^{n} n_i t_i) - (\sum_{i=1}^{n} n_i)(\sum_{i=1}^{n} t_i)}$$

The time t_s is sometimes called message latency, and the reciprocal of t_c is sometimes called the bandwidth. It is surprisingly difficult to obtain reliable estimates of t_s and t_c. For particular systems, the best thing to do is to actually write programs that send messages and take timings.

4 Speedup and Efficiency

From theoretical point of view [Pac97, JáJ92], *speedup* s and *efficiency* ξ are defined for parallel algorithms as: $s = \tau_s/\tau_p$ and $\xi = s/p$; where τ_s is the run time of the sequential algorithm and τ_p is the run time of the parallel algorithm. Let τ^* be the running time of the best known sequential algorithm to solve the problem. The *absolute speedup* s_{abs} and the *relative speedup* s_{rel} are defined as: $s_{abs} = \tau^*/\tau_p$ and $s_{rel} = \tau_1/\tau_p$; thus, the *absolute efficiency* ξ_{abs} and the *relative efficiency* ξ_{rel} as: $\xi_{abs} = s_{abs}/p$ and $\xi_{rel} = s_{rel}/p$.

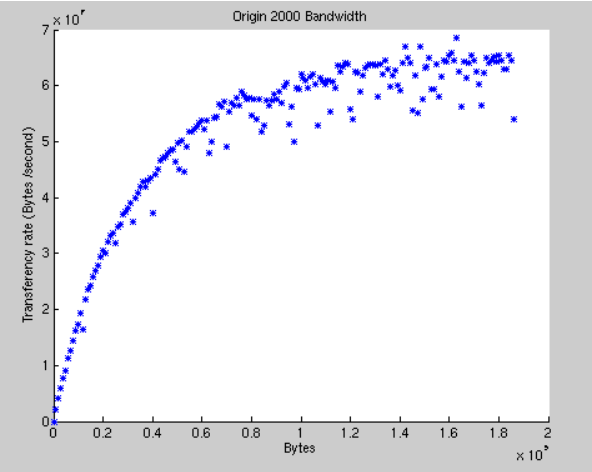

Figure 2: Bandwidth for Origin 2000

Because of $\tau^* \leq \tau_1$, $s_{abs} \leq s_{rel}$; and a certain machines and algorithms even $p \leq s_{rel}$ is possible. Such a superlinear relative speedup can be observed due the memory hierarchies and remote accesses in parallel machines. For the same reason, a superlinear absolute speedup is possible too. s_{rel} is used to demonstrate how good a certain algorithm can be parallelized. But note, that a slow and efficient algorithm often can be executed in parallel with large $s_{rel} \cdot s_{abs}$ appears in many cases to be quite low, especially for a large number of processors, limited memory in a node processor and a lot of interprocessor communication.

As application of these ideas, first let's estimate the running time of the *serial trapezoidal rule*. Let the endpoints (a and b), and the number of trapezoids (n), and then computes a running sum: it computes the area of the ith trapezoidal and adds it into the sum of the areas of previous i-1 trapezoids. The heart of the program is

```
h=(b-a)/n;
integral=(f(a)+f(b))/2.0;
x=a;
for(i = 1; i<= n-1; i++){
    x = x + h;
    integral = integral + f(x);
}
integral = integral*h;
```

The execution time on a single processor displayed in table 1 belong to π value calculation where $f(x) = 4/(1+x^2)$; $a = 0$ and $b = 1$.

Now estimate the runtime of the *p*arallel trapezoidal rule. Once again, we include the part of our parallel program that corresponds to the serial program analyzed before.

n (10^5)	1	10	50	100
Time in s	0.0224	0.224	1.123	2.245

Table 1: Execution time for a on a single processor RISC R10000

```
h=(b-a)/n;
local_n=n/p;
local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
/* Call the serial trapezoidal function*/
integral = Trap(local_a,local_b,local_n,h);
```

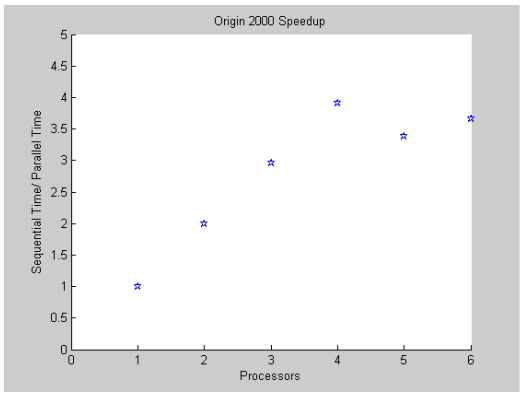

Figure 3: Speedup for parallel trapezoidal rule using $50 * 10^5$ trapezoids

5 Conclusions

In this paper we presented basic benchmark experiments for Origin 2000 with 8 processors. This benchmarks are executed successful in the batch system. In the *ping-pong* experiment we find that interchange of message size at order 10^7 take approximately 0.7 seconds, i. e. the message passing system is faster that the other parallel machines. In the bandwidth experiment the transference rate is powerful at order 10^7 bytes per seconds. In other hand we studied theoretically and practically the speedup; in order to find the speedup we used the parallel trapezoid rule for quantify the scalability. We used 10^5 trapezoids for π calculation. We found that the speedup using MPI and BSP are linear. These results can be change from one machine to another one.

6 References

References

[JáJ92] J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.

[Lau98] D. Lenoski J. Laudon. The origin 2000: A ccnuma highly scalable server. *Silicon Graphics. Inc*, 1998.

[Pac97] Peter Pacheco. *Parallel Programming with MPI*. Editorial Morgan Kauffman, 1997.

[Sai96] D. Bailey S. Saini. Hot chips for high performance computing. *In Supercomputing Tutorials*, 1996.

[Sny95] L. Snyder. Experimental validation of models of parallel computation. *J. Van Leuwen, editor, Computer Science Today. Springer LNCS 1000*, 1995.

[Val90] V. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 1990.