Scheduling to Overcome the Multi-Core Memory Bandwidth Bottleneck

Dave Field[1], Deron Johnson[1], Don Mize[1], Robert Stober[2]

[1]*Hewlett-Packard, 3000 Waterview Parkway, Richardson, TX 75080, USA*
[2]*Platform Computing, Inc., 101 Metro Drive, Suite 540, San Jose, CA. 95110*

Platform™

Abstract

Need more compute servers but don't have room for them? Or maybe you don't have enough electrical capacity to feed them, or enough cooling capacity? You're not alone. Multi-core processors may be the solution.

However, while multi-core processors may solve many of the problems associated with compute cluster sprawl, they also present a new challenge: in some situations they cannot provide sufficient memory bandwidth per core to satisfy the requirements of certain HPC applications.

This paper discusses methods to mitigate the effects of memory bandwidth limitations on modern multi-core processors.

Introduction

Multi-core processors promise to deliver significant performance benefits to the enterprise while relieving some of the pain that has accompanied the compute cluster sprawl. Each core contains its own set of execution resources resulting in very low latency parallel execution of application threads within a single physical CPU package.

The benefits of multi-core processors are not limited to increased performance. Multi-core processors provide greater system density, allowing organizations to maximize the productivity of their available floor space. Since they operate at lower frequencies, multi-core processors use less power and generate less heat per core than the commensurate number of single-core processors.

Multi-core processors do, however, present a new challenge that will need to be met if they are to live up to expectations. Since multiple cores are most efficiently used (and cost effective) when each is executing one thread, organizations will likely want to run one job per core. But many of today's multi-core processors share the front side bus as well as the last level of cache. Because of this, it's entirely possible for one memory-intensive job to saturate the shared memory bus resulting in degraded performance for all the jobs running on that processor.

The front side bus, which is also known as the memory bus, is the "highway" upon which data travels as it is written to or read from memory. "Memory bandwidth" is the amount of data that can travel on the memory bus in a given period of time, and is usually expressed in units of MB/sec. Although improvements in memory system performance have historically lagged behind improvements in processor performance, the chip manufacturers are working hard to close the gap. But even if they're successful, if the new multi-core chips implement significantly faster memory systems, as long as the memory bandwidth is shared between the cores there will always exist the potential for bottlenecks. And as the number of cores per processor and the number of threaded applications increase, the performance of more and more applications will be limited by the processor's memory bandwidth.

The Solution

One technique which mitigates this limitation is to intelligently schedule jobs onto these processors, managing the memory bandwidth demand versus its supply. Platform LSF®[1] can be configured to automate this technique.

A Platform LSF "resource" that represents the amount of available memory bandwidth is created and assigned to each compute server. The value of the resource can be configured on a host-by-host basis, and can easily be determined using the STREAM standard benchmark. The memory bandwidth resource is shared among the jobs running on the server.

When a job is submitted specifying its memory bandwidth requirement, Platform LSF will automatically dispatch the job to a server that has enough memory bandwidth to meet the requirement, and will maintain each server's memory bandwidth balance as jobs are scheduled and completed. Since the memory bandwidth resource can be implemented within the Platform LSF configuration files, customers can implement and use the feature with their existing Platform LSF version.

Configuration

1. Determine the bandwidth of each of your compute servers.

Maximum CPU memory bandwidth may be obtained from the vendor's technical documentation or by using the STREAM[2] standard benchmark. The STREAM benchmark and

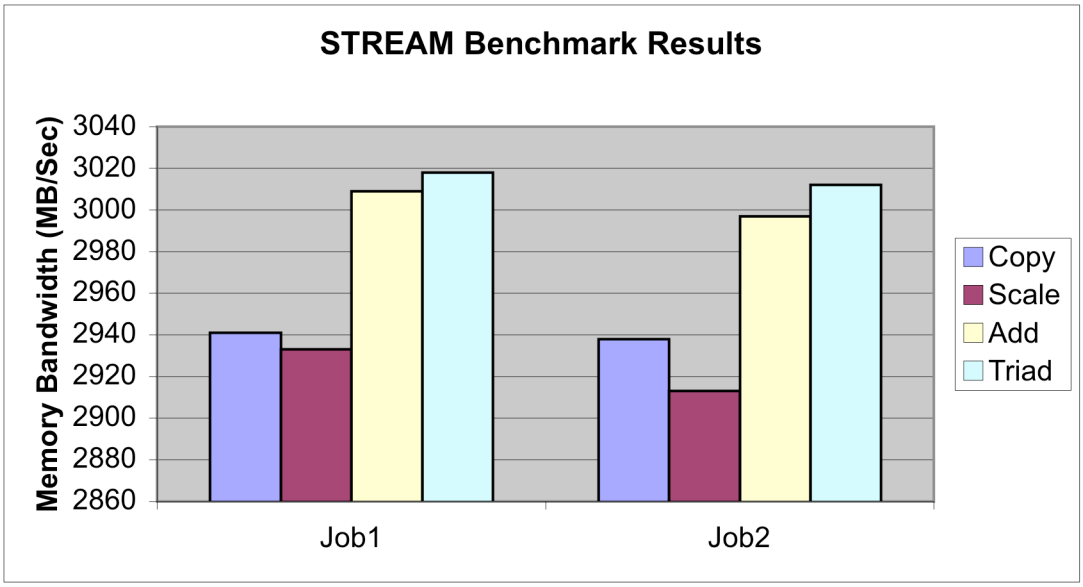
[1] Platform LSF is software for managing and accelerating batch workload processing for compute-and data-intensive applications. See www.platform.com

[2] McCalpin, John D., 1995: "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

additional information can be obtained from the STREAM benchmark home page at http://www.streambench.org/.

The following chart shows the results of running two simultaneous STREAM instances on an HP ProLiant server containing two dual-core processors. By adding together the "Triad" results, we can see that this processor can provide about 6,000 MB/sec of memory BW.

Figure 1. Bandwidth of two dual-core processors as determined by the STREAM benchmark.



2. Create the LSF memory bandwidth resource

```
# Edit the lsf.shared file

# Add a new type of resource that will be associated with all of
# the compute nodes managed by LSF. The name of the resource is
# "bw", but you can name it as desired.

Begin Resource
RESOURCENAME  TYPE    INTERVAL INCREASING  DESCRIPTION
bw            Numeric ()       N           (available bandwidth)
End Resource
```

3. Initialize the Platform LSF memory bandwidth resource on each server

Configure the initial value of the resource to the total memory bandwidth of each system. For example, if each processor of a dual-processor system has 6,000 MB/sec memory BW, we would set the initial value of the resource to 12,000 for that system.

```
# Edit the lsf.cluster.clustername file

# Now that the new resource type for memory bandwidth "bw" has
# been added, we need to edit the lsf.cluster.clustername file to
# add in the specific values of bw for the different types of
# hardware in the cluster. Note that hosts or host groups may be
# specified.

Begin ResourceMap
```

```
RESOURCENAME  LOCATION
bw            (1500@[hostA] 12000@[hostB] 6000@[hostC])
End ResourceMap
```

4. Reconfigure the LSF cluster to apply the changes

```
> lsadmin reconfig
> badmin reconfig
```

5. Submit jobs specifying the BW resource requirement

The following command submits the job "myjob" to the default Platform LSF queue. The job will be dispatched to a compute server having at least 800 MB/sec of available bandwidth. And once dispatched, that amount of memory bandwidth will be reserved for the duration of the job.

```
bsub –o stdout –R "select[bw>800] rusage[bw=800]" myjob
```

The bhosts –s command can be used to display the value of the resource on each server.

```
$ bhosts -s
RESOURCE                 TOTAL         RESERVED       LOCATION
bw                       11200.0          800.0       hostB
bw                        1500.0            0.0       hostA
bw                        6000.0            0.0       hostC
```

6. Configure (optional) default BW resource requirement

Default resource requirements can be configured on the queue or application level. They take effect when the user does not specify the resource on the command line. To configure a default resource requirement string, add the RES_REQ parameter to the lsb.queues or lsb.applications file.

```
RES_REQ = select[bw>500] rusage[bw=500]
```

Results

We selected the STREAM benchmark to be the test application to measure the performance of the new Platform LSF resource. We ran STREAM using one thread. In effect, we implemented a batch stream, which produces the fastest time-to-completion of memory-BW-intensive codes.

We tested two configurations—HP ProLiant servers using dual-core processors and HP ProLiant servers using quad-core processors.

Dual Core

We used HP ProLiant DL140 servers, each containing two Intel 5140 dual-core processors. This server has a total of 4 cores.

We started by submitting four STREAM jobs and configured the memory BW resource such that each job required 40% of the total server BW. Platform LSF dispatched the jobs to an execution server two at a time, since a third job would consume more BW than our limit. Each dual-core processor ran one STREAM instance, which only consumed about 2/3 of its available memory BW.

Next, we submitted the four STREAM jobs and disabled the BW resource, by configuring the memory BW resource such that each job required < 25% of the total server BW. Platform LSF dispatched all four jobs at once. Each dual-core processor ran two STREAM jobs simultaneously, which required about 1/3 more memory BW than was available. As a result, the average runtime of each job increased to 1.87 times longer than the job runtime using the BW resource.

Quad-Core

We used an HP ProLiant DL140 G3 server, containing two Intel 5355 quad-core processors. This server has a total of 8 cores.

We submitted eight STREAM jobs and configured the memory BW resource such that each job required 40% of the total server bandwidth. As in the dual-core test, Platform LSF scheduled only 2 jobs at a time on the server, using 1/4 of the cores on the server.

Then, we disabled the BW resource and submitted the eight STREAM jobs to this server. Platform LSF dispatched all eight jobs at once. Each quad-core processor ran four STREAM jobs, greatly exceeding the available memory BW of the processor. As a result, the average runtime of each job increased to 3.47 times longer than the job runtime using the BW resource. Among a series of jobs, this runtime ratio varied from 3.45X to 3.49X.

Conclusion

These test results clearly demonstrate that Platform LSF can be used to schedule jobs based on available memory bandwidth. How the Platform LSF resource should be configured depends entirely on your goal. Are you trying to obtain the shortest possible turn-around time for high priority jobs, or are you trying to get the highest throughput possible?

If the goal is the shortest possible turn-around time for specific jobs, the Platform LSF resource should be set so that the application is not limited by memory BW. This approach is necessary for high-priority jobs, when job time-to-completion must be minimized. Also, it makes effective use of application licenses, minimizing the time that each job utilizes these licenses.

If, on the other hand, the goal is to maximize cluster throughput, the Platform LSF resource should be set higher; slightly over-subscribing a server's available memory bandwidth results in a shorter turn-around time *for a set of jobs* than does under-subscribing. In this case, individual jobs will take longer to complete, but because there are more jobs running simultaneously the throughput is maximized.

Both of these scheduling objectives can be accommodated within a single cluster by assigning hosts configured with a relatively low Platform LSF BW threshold to a high priority queue and others with relatively high Platform LSF BW threshold to a high throughput queue.

Acknowledgments

The idea for this project originated in HP's High Performance Computing Division. It is one of the results of HP's Multi-Core Optimization Program, which seeks ways to improve total application performance and per-core application performance on servers using multi-core processors.

For more information

www.hp.com/go/hpc
Hewlett-Packard High Performance Computing home site

www.platform.com
Platform Computing home site

4AA1-6088ENW, November 2007

Platform

hp invent