Optimizing GPU-based application performance for the HP ProLiant SL390s G7 server

Technology brief

Introduction ................................................................. 2
GPU-based computing .................................................... 2
ProLiant SL390s GPU-enabled architecture .................................. 2
Optimizing GPU applications for maximum bandwidth .................................. 3
  Local vs. remote CPU/Memory ........................................ 3
  Setting the processor and GPU for an application ....................... 4
  Locating GPUs in the SL390s .......................................... 5
  Optimizing bandwidth using the bandwidthTest in the CUDA SDK .................. 5
Additional system optimizations ........................................ 7
  QPI bandwidth optimization ........................................... 7
  Memory node interleaving .................................................. 7
  Use of Quad-Core vs. Six-Core processors .................................. 8
Conclusion ........................................................................ 8
For more information ......................................................... 9
Introduction

Enterprise-level customers are showing increased interest in High Performance Computing using Graphic Processing Units (GPUs) in industry standard servers. To address this growing market, HP has introduced the ProLiant SL390s G7 2U server—part of the ProLiant SL6500 Scalable System family. We designed the SL390s specifically for GPU-based computing, and it supports up to 3 GPUs in its 2U half-width tray. This technology brief gives you an overview of the unique architectural features of the SL390s that support high-density, high-performance GPU-based computing. It also provides insight and guidance for designing and tuning GPU-based applications for maximum performance on the SL390s.

GPU-based computing

GPU-based computing has its beginnings in the Graphic Processing Units (GPUs) created for industry standard computers in the 1990s. GPUs originally performed the complex operations required to render 3D graphics in real time. Over time, computer scientists and the companies that produced GPUs realized that they could harness the large-scale parallel-computing capability of GPUs to run general-purpose, computational applications. The industry originally referred to this application area as GPGPU, for general-purpose computing on Graphic Processing Units.

Today, leading vendors have created specialized GPU modules as well as development that make them fully programmable for computational applications. The Tesla GPU modules supported in the SL390s use the NVIDIA® GPU platform and run compute applications developed using the CUDA or OpenCL programming model and software development tools.

The basic work units of GPU computing are floating-point mathematical operations. Current GPU modules can execute hundreds of gigaflops (1 gigaflop is a unit equal to 1 billion floating-point operations per second), nearly 10 times the floating-point performance of current general-purpose quad core processors. The SL390s, with three GPUs, can achieve greater than 1 teraflop (1 trillion flops) double-precision performance on the Linpack benchmark.

ProLiant SL390s GPU-enabled architecture

Attaining the best performance for GPU-enabled applications on server systems requires maximizing the bandwidth between system memory and the GPU modules. GPU-based computing typically involves moving very large data sets and results sets back and forth between system memory and the GPU over the PCIe bus. Our engineers have designed the SL390s to provide maximum PCIe bandwidth for the GPUs in the system. The SL390s architecture incorporates two PCI I/O Hubs, providing 64 PCIe 2.0 lanes for system I/O. PCIe lanes are used by only one PCIe slot, with separate x16 PCIe slots for each of the 3 GPUs that the system supports. As a result, the system does not use I/O splitters, which reduce performance by requiring two PCIe slots to share PCIe lanes. Figure 1 is a simplified block diagram of the core systems architecture showing the key I/O interconnects, including the QuickPath Interconnect (QPI) links that provide high-speed data transport between the Intel Xeon® processors and the I/O Hubs.
Figure 1. The ProLiant SL390s core architecture features 2 I/O Hubs.

Optimizing GPU applications for maximum bandwidth

Today, most customers develop their own GPU-enabled applications using the programming tools for that particular platform. Because most GPU-enabled applications are unique, we cannot provide detailed performance tuning information for every situation. However, we can provide general performance and configuration guidelines that should help you develop and tune your applications to achieve optimal performance.

Local vs. remote CPU/Memory

Today’s NUMA architected systems split system memory, with different parts of the memory physically attached to different processors. CPU 0 can access the portion of the memory attached to it directly and more quickly than the memory attached to CPU 1, which requires access across the QPI (QuickPath Interconnect) bus. With the dual I/O Hub architecture of the SL390s, this extends to the I/O system, because you can attach each CPU and GPU directly to one of the I/O Hubs and not to the other.

As a result, each GPU in the system has a shorter path to one CPU/memory complex than to the other. This arrangement allows the GPU to move data to and from its attached memory much quicker. As Figure 2 illustrates, GPU 0 can perform I/O to and from CPU 0 and its memory directly through I/O Hub 0. This is the local path between the GPU and system memory. Transferring data to and from CPU 1 requires going through both I/O Hubs, resulting in lower I/O bandwidth. Similarly, GPU 1 and GPU 2 will achieve maximum bandwidth when transferring data sets to and from the memory attached to CPU 1.
As a general guideline, the best way to optimize the performance of a GPU-enabled application is to maximize the I/O bandwidth between the GPUs and memory by running applications on CPU/GPU pairs that have a local path. Fortunately, Linux and most GPU-enable applications have methods allowing you to do just that.

Setting the processor and GPU for an application.

Linux provides two commands that specify which processor core runs a particular task—the `taskset` command and the `numactl` command. On the SL390s with the Intel Xeon 5600 series processors, even core numbers are located on CPU 0 while odd core numbers are on CPU 1. Knowing this, we see that all of the following commands specify running a process called `yourgpuapp` on CPU 0.

```
taskset -c 2 yourgpuapp
 taskset -c 4 yourgpuapp
 numactl -physcpubind=2 yourgpuapp
```

To optimize the GPU for memory bandwidth, you should also ensure that the application uses the appropriate GPU. This is usually done through a command line switch for the application. For the following example to work, the developer must have designed `yourgpuapp` to be able to take the `device=X` command line switch to designate the GPU to use.

```
taskset -c 2 yourgpuapp --device=0
```

Using this Linux command line, the system will execute `yourgpuapp` on CPU 0 using GPU 0, ensuring a local path between GPU and memory and delivering the best performance.
For more information on using `taskset` and `numactl`, please refer to the Linux `man` pages.

**Locating GPUs in the SL390s**

Because the mapping of GPUs to CPUs is significant in GPU-based computing, you need to install the GPUs in the correct locations, especially if you are not installing the maximum of 3 GPUs. Figure 3 shows the physical locations of the three different GPUs in a fully configured SL390s chassis and their CUDA device numbers.

*Figure 3.* The GPUs in the ProLiant SL390s 2U server are physically located in upper and lower trays.

If you are only installing a single GPU, then you should install it in the lower tray. This ensures that GPU 0 has a local data path to CPU 0. Similarly, if you configure the system with 2 GPUs, one should be in the upper tray and the other in the lower one. With this configuration, each of the GPUs has a local path to a CPU.

**Optimizing bandwidth using the bandwidthTest in the CUDA SDK**

The `bandwidthTest` in the CUDA SDK is one of the most common programs used to check GPU bandwidth performance. It also illustrates how you use the `taskset` command to ensure that tasks execute using the proper CPU/GPU combination to maximize bandwidth performance.

The following set of commands executes the `bandwidthTest` on each of the 3 CPU/GPU pairings in the SL390s, using the local CPU/GPU I/O path in each case.

```
# CPU0+GPU0
taskset -c 2 ./bandwidthTest --memory=pinned --device=0
```
The bandwidthTest measures the ability to move data, in the form of input datasets or results sets, within the system. It can measure three different bandwidth metrics:

- Device to Host Bandwidth – that is, from the GPU to the CPU/system memory
- Host to Device Bandwidth – that is, from the CPU/system memory to the GPU
- Device to Device Bandwidth – moving data within a single GPU’s memory

The following illustrates the results from running a BandwidthTest command for GPU 0.

```
./bandwidthTest -memory=pinned -device=0
./bandwidthTest Starting...
Running on...
Device 0: Tesla M2050
Quick Mode
Host to Device Bandwidth, 1 Device(s), Pinned memory, Write-Combined Memory Enabled
  Transfer Size (Bytes)   Bandwidth(MB/s)
  33554432               5747.5
Device to Host Bandwidth, 1 Device(s), Pinned memory, Write-Combined Memory Enabled
  Transfer Size (Bytes)   Bandwidth(MB/s)
  33554432               6204.2
Device to Device Bandwidth, 1 Device(s)
  Transfer Size (Bytes)   Bandwidth(MB/s)
  33554432               86365.5
```

[bandwidthTest] - Test results:
PASSED

The bandwidthTest should produce the best results when you map the CPU/GPU combinations using the local paths as we have described. In the example above, the test shows a 5.7 GB/s transfer rate from Host CPU to the GPU and a similar 6.2 GB/s transfer rate from the GPU to the Host CPU. As we would expect, the transfer rate of 86 GB/s for moving the data within the GPU itself is extremely fast since the PCIe bus is not involved.

GPU/CPU bandwidth is one factor that can affect the performance of GPU-enabled applications. If the system does not have enough bandwidth to move data and results sets quickly and efficiently between the system memory attached to the CPUs and the GPU’s memory, then overall performance is constrained. However, if enough bandwidth is available for the application to perform these transfers efficiently, then more bandwidth will not significantly increase performance. In this situation, the ability of the GPU(s) to process the data becomes the primary influencer of performance.
Additional system optimizations

Although proper mapping of CPUs to GPUs is one of the most important steps you can take to optimize system performance, there are several other configuration settings and choices that you should consider when tuning the system for your GPU-enabled application.

QPI bandwidth optimization

In the Intel NUMA architectures, the QPI performs two distinct functions—moving data between system memory and the CPUs and moving data between memory and I/O device through the I/O Hub and the PCIe bus. The system has to allocate the I/O Hubs’ and QPI’s total resources against these 2 tasks.

With the Intel Xeon 5600 series processors used in the SL390s, you can configure the allocation of these resources between processors, memory, and I/O. The ROM-Based Setup Utility (RBSU) features a new Advanced Performance Tuning option called QPI Bandwidth Optimization. You usually set it to Balanced mode, the recommended setting for the majority of general-purpose computing. Setting the QPI Bandwidth Optimization to Optimized for I/O allocates more resources to I/O transfers. This can improve the device-to-host bandwidth for the GPU1 and GPU2, which share I/O Hub 1. It can also improve the bi-directional transfer bandwidth for all of the GPUs. GPU-based applications, which tend to be very I/O intensive, may achieve improved performance with this setting.

Memory node interleaving

In a NUMA environment, separate memory is attached to each processor, or node, in the system. You can alter the way in which the system translates physical memory to the logical system memory map using Memory Node Interleaving setting in RBSU. By default, the system has Memory Node Interleaving Off.

Each set of physical memory attached to a processor node which is then a contiguous block in the system memory map. For most environments, including most GPU-based computing, this is the proper setting. It normally delivers the highest I/O bandwidth from CPU to GPU when the architecture of the application uses the local CPU to GPU paths.

When you set memory node interleaving to On, the location of each 4k page of the system memory map alternates between the different processors’ physical memory. As a result, data set I/O transfers from memory to a given GPU will also alternate between 2 different data paths. Turning memory node interleaving on will deliver more consistent (although usually lower) bandwidth for I/O operations across all GPUs regardless of the CPU/GPU pairing used.

There may be an advantage to turning Memory node interleaving On for those applications that load data sets large enough to span the physical memory of both CPU sockets and that use all GPUs.

Having interleaving On will minimize the performance difference between accessing the near and far memory and could potentially deliver better overall performance in these cases.

You can also turn interleaving on using the Linux numactl command when running your GPU-enabled application from the command line.

```
Numactl --interleaving=all yourgpusapp
```

Testing has shown that using numactl for interleaving produces different performance results than turning memory node interleaving on in the BIOS. If you have the type of application that might benefit from interleaving, it’s best to investigate both alternatives.
Use of Quad-Core vs. Six-Core processors

The SL390s supports both Quad-Core and Six Core Intel Xeon 5600 series processors. In general-purpose computing applications, servers equipped with Six-Core processors will outperform those with Quad-Core processors because the system can distribute program threads across a greater number of cores. In GPU applications environments, the answer may not be so straightforward.

In application development environments such as CUDA, you can design an application to utilize any processing resources in the system, including the CPUs as well as the GPUs. Applications that are more complex may use the system processors to work smaller pieces of the overall problem while sending larger pieces to the GPUs. They may also use the system processors to consolidate work performed using the GPUs.

Overall, GPU-enabled applications that make extensive use of the systems processors and multi-threading may benefit from a system with Six-Core processors. Applications that deal in large data sets and primarily use the GPU’s will probably do fine using Quad-Core processors. As always, the only way that you can know for sure is to test the applications that you have developed or intend to use on the servers.

Conclusion

The ProLiant SL390s G7 represents a new category of industry-standard servers designed specifically around enabling GPU-based computing. With this architecture, HP is delivering compute-intensive solutions that have 10 to 20 times the performance per dollar of traditional CPU-based solutions as well as leaders in compute density and performance power efficiency. The high end of GPU-based computing continues to be a somewhat specialized market, with most GPU-enabled applications developed by the customers to solve specific compute-intensive scientific and engineering problems. The information presented in this document provides some general guidelines for optimizing performance of the GPUs in the SL390s. Because of the unique nature of most GPU-enabled applications, detailed performance tuning may also require a good working knowledge of the architecture of the individual applications in addition to the architectures of the SL390 and GPUs.
For more information

Visit the URLs listed below if you need additional information.

<table>
<thead>
<tr>
<th>Resource description</th>
<th>Web address</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP High Performance Computing</td>
<td><a href="http://www.hp.com/go/hpc">www.hp.com/go/hpc</a></td>
</tr>
<tr>
<td>HP Accelerator Program</td>
<td><a href="http://www.hp.com/go/accelerators">www.hp.com/go/accelerators</a></td>
</tr>
</tbody>
</table>

Send comments about this paper to TechCom@HP.com

Follow us on Twitter: http://twitter.com/ISSGeekatHP

© Copyright 2011 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Tesla is a trademark and NVIDIA is a registered trademark of NVIDIA Corporation.

Intel, and Intel Xeon are trademarks of Intel Corporation in the United States and other countries.

TC1001728, September 2011