“Large Matrix-Matrix Multiply on PS3 clusters”

Dennis Fitzgerald, ITT dennis.fitzgerald@itt.com
Mark Barnell, AFRL mark.barnell@afrl.mil

Large dense matrix-matrix multiplication is a computationally intense algorithm that presents performance issues associated with managing cache memory while moving large amounts of data. Data bandwidth can also be a problem when parallelizing the algorithm over a number of processors. This paper discusses a parallel implementation of matrix-matrix multiply using a cluster of Sony PlayStation 3 Linux workstations. The PS3 is based on a version of the IBM Cell BE processor that is capable of performing 153 GFLOPS in single precession. The memory constraints of the PS3 (256MB of main memory and 256KB on each Synergistic Processing Element (SPE)) limit the amount of data that can be processed in each step. However, there has been work performed by Daniel Hackenberg [1] demonstrating 99.43 percent of the theoretical peak performance of the PS3, for small matrix multiplies. These matrices must be square, fit in the PS3’s main memory and have dimensions being a multiple of 128. To apply a cluster of PS3s to the problem of solving large matrix-matrix multiply, the matrices must be blocked in such a way that maximizes IO bandwidth while using a limited memory size. This approach keeps the SPEs busy while minimizing the amount of time lost due to overhead. Using the combined bandwidth of the 1Gb Ethernet connection to the PS3 and the local disk storage, this paper discusses how the available IO bandwidth, appropriate data sizing and block processing method can be sufficient to allow the PS3 cluster to perform a large matrix-matrix multiplication while maintaining very high percentage of peak performance on each processor.

Miriam Leeser of Northeastern University used the Dresden code as a core to produce a program that multiplies rectangular matrices while maintaining nearly the same high level of performance as the original code. The limitation of the matrix sizes is based on the 256 MB of PS3 memory. This code was then used on multiple PS3s in combination with a Xeon based head-node to perform matrix-matrix multiplication of very large matrices which could be dimensioned in millions of elements.

To take advantage of all available IO, the A matrix data is read from the PS3’s local disk and the B matrix data is transmitted from the head-node through the PS3’s 1Gb Ethernet network connection. Both disk and network IO is handled by the PS3’s SPU which allows overlapping of the IO with the computations being performed on the SPEs. The disk and network IOs are also overlapped through the use of multiple threads handling each IO, maximizing throughput through the use of multiple buffers.

To maximize performance, the compute time per unit of work must closely match the IO time required to deliver the amount of data for the next compute cycle. In this case, rectangular matrices of 4K x 128 (A matrix) and 128 x 8k (B matrix) were selected because the network bandwidth is approximately twice local disk bandwidth. This approach equalizes the delivery time of the two matrices, maximizing the combined IO throughput. In addition, the IO time of these two matrices closely matches the combined compute time of the 6 SPUs to perform the multiplication. Finally, the available memory is sufficient to hold two buffers for each A and B matrix and the C product matrix.

A matrix - 4K X 128 X 4(size of single precision) X 2 buffers = 4,194,304 bytes
B matrix - 8K X 128 X 4(size of single precision) X 2 buffers = 8,388,608 bytes
C matrix - 4K X 8K X 4(size of single precision) = 134,217,728 bytes
Total = 140 MB

To maximize memory performance huge pages are used to minimize the delay caused by translation lookaside buffer (TLB) misses. The size of a huge page is 16MB which means that 9 huge pages are required to address all buffers.

Since the target matrix size is greater than 4k or 8k, larger matrices are multiplied using a blocked method allowing the large matrices to be multiplied in smaller “blocks”. Figure 1 illustrates how a larger set of matrices (48k x 48k in this case) would be blocked. Each PS3 has one blocked row (4k X 48k) stored on its local disk which is read into memory one block (4k X 128) at a time. The head-node transmits blocked B columns to all PS3s one block (128 x 8k) at a time. Each block multiplication along the A blocked row and B blocked column is accumulated in the resulting 4k x 8k C matrix. Each PS3 computes a different block of the product C matrix corresponding to the intersection of the A blocked rows and B blocked columns that that PS3 is processing. The A matrix blocked row is reused for each B blocked column to produce another block of the C matrix which is stored on the local PS3 disk.

The scalability graph, Figure 2 displays the performance of a 48k x 48k matrix multiply and a 48k x 240k case. The improved scalability of the 48k x 240k case is due to the fact that the more operations are performed and summed into the C sub-matrix before the C matrix needs to be saved. Since the resulting C sub-matrix is the same size in both cases. The relative amount of time required to save it is reduced.

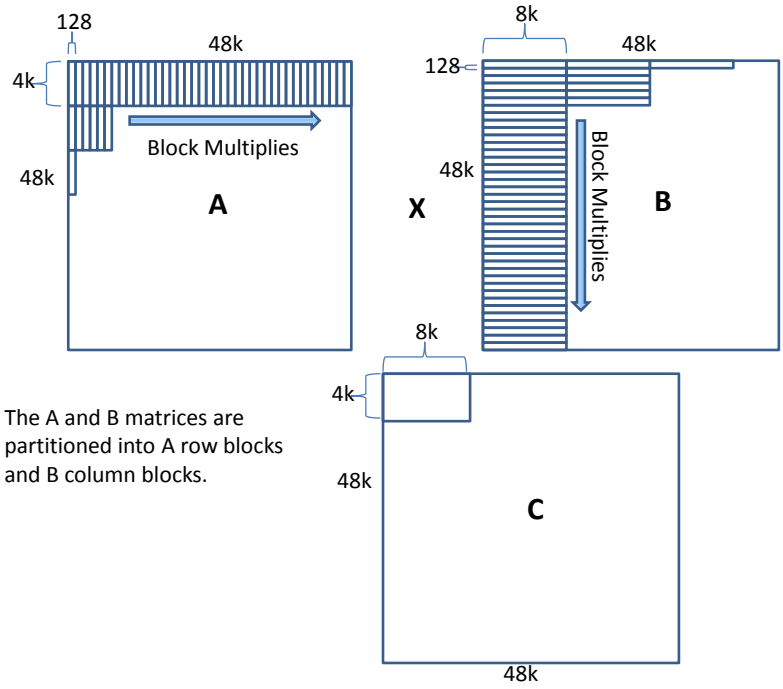

Figure 1: Matrix Partitioning

The same performance data can be displayed as an average performance per PS3, Figure 3. By displaying the data in this format, the decreasing number of FLOPS is easier to see. This graph was found to be useful when working to improve the cluster performance. In this case the drop off is attributed to network IO. However, tests performed solely on network IO bandwidth indicated that the drop-off should not have been this severe. Further work is needed to better understand the IO performance in this case

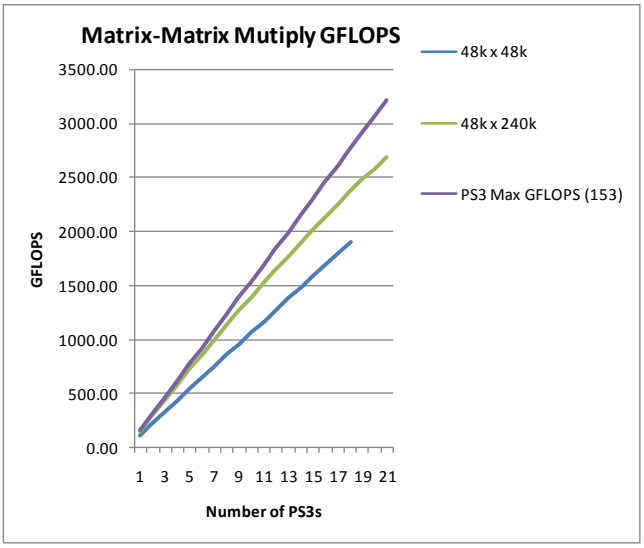

Figure 2: Scalability Graph

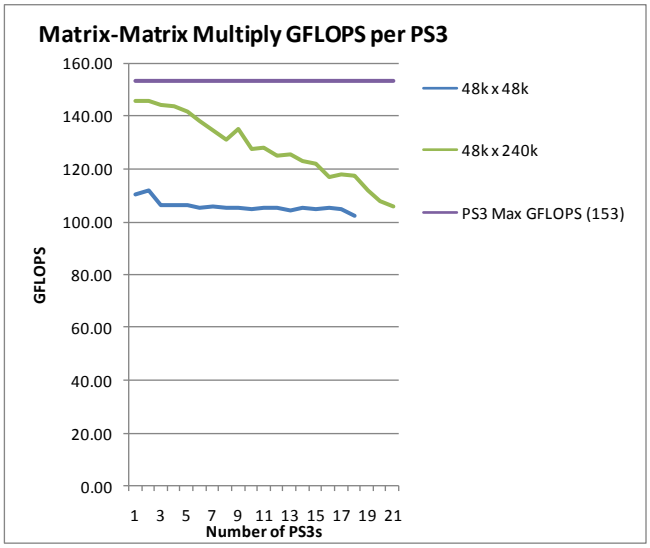

Figure 3: Average Performance per PS3

References

[1] Daniel Hackenberg, TU-Dresden, 2007, "Fast Matrix Multiplication on Cell (SMP) Systems", http://www.tu-dresden.de/zih/cell/matmul.

[2] Richard Linderman, "Early Experiences with Algorithm Optimizations on Clusters of Playstation 3's", DoD HPCMP Users Group Conference, 2008.