Computer Architecture: Project 4

Aniket,Deepak,Sagar,Shruti

Answer Part 1:

Configuration: In common.h, Array Size set as 128. (Computation on PPU)
Computation Time: 104364 us
Solution:

Multiplication of two 128*128 matrix was done on PPU itself, with Normal Steps of Multiplication. Matrix Multiplication result was also printed on PPU itself.

Answer Part 2:

Configuration: In common.h, Array Size set as 128. (Computation on 8 SPU)
Computation Time: 310578 us (For small Arrays time consumed in context switch and DMA transfer)
Solution:

Multiplication of two 128*128 matrix was done by dividing the task among 8 SPUs. We created 8 threads for each SPU and then computed part of the multiplication on each SPU. We sent 16 rows from the first matrix and 16 columns from the second matrix to each SPU. We achieved this by taking a transpose of second matrix before Multiplication, so that all column elements are at sequential memory locations to ease DMA operation. We split the 128*128 matrix in to group of 16 rows and 16 columns, thus computing a 16*16 matrix i.e 256 elements on each SPU. This can be further parallelized to compute 32 rows and 32 columns, i.e a 32*32 matrix on a single SPU. While doing this we made sure that the result multiplication matrix on each SPU is small enough to meet SPE memory requirement.

Answer Part 3:

Configuration: In common.h, Array Size set as 1024. (Computation on 8 SPU, repetitive call to SPU)
Computation Time: 8137512 us
Solution:

Multiplication of two 1024*1024 matrix was done by dividing the task among 8 SPUs. Our approach was to compute as many elements as possible within one SPE context run. So we initially passed 16 rows from first 1024*1024 Matrix and 16 columns from second 1024*1024 Matrix (taking transpose) to each SPU context run. The rows and column combination sent to each SPU was different. We thus computed 8, 16*16 Matrix from 8 SPU threads in Parallel. Transpose of second matrix before Multiplication was taken, so that all column elements are at sequential memory locations to ease DMA operation. We split the 1024*1024 matrix in to group of 16 rows and 16 columns, thus computing a 16*16 matrix i.e 256 elements on each SPU. We needed 64*64 such computations in order to compute the entire multiplication matrix.

Answer Part 4: Parallelize as much as possible.

Configuration: In common.h, Array Size set as 1024. (Parallelize till SPU memory requirements are met)
Computation Time: 8063128 us

Memory on SPU:

We tried to parallelize the 1024*1024 Matrix computations by passing 32 rows and 32 columns to each SPU, trying to compute 1024 elements on one SPU in one go. However, considering the memory requirements of SPU, We needed a total storage of

Int Mat1[32][1024] (131072 bytes)
Int Mat2[32][1024] (131072 bytes)
Int Mult[32][32] (4096 bytes)

Computer Architecture: Project 4

Aniket,Deepak,Sagar,Shruti

Total Storage needed = 266240 bytes = **266kB**
However, Each SPU can support only 256KB of memory, hence to compute 32*32 i.e 1024 elements in one go was not possible.

So, we decided to go ahead with 16*16 parallel computation approach by computing 256 elements on each SPU in one go. For this we had to pass 16 rows from the first 1024*1024 Matrix and 16 columns from second 1024*1024, thus needing to allocate below space on each SPU.
Int Mat1[16][1024] (65536 bytes)
Int Mat2[16][1024] (65536 bytes)
Int Mult[16][16] (1024 bytes)
Total storage need (Including Local variables) = **140kB** (approx.)
Which is less than 256kB available on a SPU.

Solution:
Our Solution thus aims at computing 16*16 i.e 256 elements on multiple SPUs in Parallel. As we have 8 SPU available we were able to compute 256*8=2048 elements in one parallel iteration. To compute the entire 1024*1024 Matrix multiplication we needed 512 such parallel iterations.
The smaller chunks of multiplication computed by each SPU, were merged and printed in PPU.