Parallel Computing Architectures
048874

Prof. Ran Ginosar

Sandeep Hans and Tauseef Hussain
Technion
Outline

1 Introduction
   - Plural Architecture
   - GEMM

2 Results
   - Parallel GEMM
   - Fast GEMM
   - Comparison

3 Observations
Many-core Plural architecture

- Shared Memory
- RISC Architecture
- Hardware synchronizer
- \#cores = 16, 32, 64, 128, 256
- 4-64 Coprocessors
The General Matrix Multiply (GEMM) is a subroutine in the basic linear algebra programs which performs matrix multiplication.

The GEMM routine calculates the new value of matrix $C$ based on the matrix-product of matrices $A$ and $B$, and the old value of matrix $C$ ($\alpha$ and $\beta$ are scalars).

$$C \leftarrow \alpha AB + \beta C$$
• We tried two algorithms for parallelizing GEMM
  • Parallel GEMM
  • Fast GEMM
Parallel GEMM

- Divide the computation in two parts, and then join the parts
  1. Compute dot product: \( P \leftarrow \alpha \cdot A \cdot B \)
  2. Compute scalar product: \( Q \leftarrow \beta \cdot C \)
  3. Addition: \( C \leftarrow P + Q \)
Fast Gemm

- Directly compute for every element.
  \[ C \leftarrow \alpha AB + \beta C \]
- Multiplication and addition in the same task.
- No additional dependancies.
- No extra space required.
Outline

1. Introduction
   - Plural Architecture
   - GEMM

2. Results
   - Parallel GEMM
   - Fast GEMM
   - Comparison

3. Observations
We got results for parallel and fast gemm algorithms.

- \# Cores = 16, 32, 64, 128, 256
- Size of Arrays = 2, 4, 8, 16, 32, 64, 128
Metrics

- Speed Up for $X$ cores: execution time with $X$ cores w.r.t. execution time with base cores.
- Speed up per core = \( \frac{\text{Speed Up for } X \text{ cores}}{X} \).
Parallel GEMM

- For computing $C$:
  1. $N \times N$ task with $N$ multiplications each for dot product
     \[ P[i][j] \leftarrow \alpha \cdot \sum_{k=0}^{N-1} A[i][k] \cdot B^T[i][k] \]
  2. $N \times N$ tasks for scalar product
     \[ Q[i][j] \leftarrow \beta \cdot C[i][j] \]
  3. $N \times N$ task for addition
     \[ C[i][j] \leftarrow P[i][j] + Q[i][j] \]

- Task 1 and 2 can run in parallel.
- Task 3 is dependent on both Task 1 and Task 2.
Task Map for Parallel GEMM

- regular task `program_init()`
- duplicable task `scalarDotProd(program_init/1)`
- duplicable task `scalarProd(program_init/1)`
- duplicable task `addition(scalarDotProd & scalarProd)`
- regular task `program_finish(addition)`
Parallel GEMM

Speed up of Parallel GEMM w.r.t. #cores
Parallel GEMM

Speed up w.r.t. size of arrays

Sandeep Hans and Tauseef Hussain

Parallel Computing Architectures - 048874
Parallel GEMM

Speed up for small arrays

Sandeep Hans and Tauseef Hussain
Parallel GEMM

Speed up per core
Fast Gemm

- Directly compute for every element
- For computing $C[i][j]$:
  - $N \times N$ tasks
  $$C[i][j] \leftarrow \alpha \cdot \sum_{k=0}^{N-1} A[i][k] \cdot B^T[i][k] + \beta \cdot C[i][j]$$
- No dependancies and no extra space required.
- But every task has a lot of work to do.
Task Map for Fast GEMM

- regular task program_init()
- duplicable task fastGemm(program_init/1)
- regular task program_finish(fastGemm)
Fast GEMM

Speed up of Fast GEMM w.r.t. #cores
Fast GEMM

Speed up w.r.t. size of arrays
Fast GEMM

Speed up for small arrays
Fast GEMM

Speed up per core
Comparison of Parallel and Fast GEMM

Speed up for Parallel and Fast GEMM for $N = 128$
Outline

1. Introduction
   - Plural Architecture
   - GEMM

2. Results
   - Parallel GEMM
   - Fast GEMM
   - Comparison

3. Observations
Observations

1. Speed up increases with number of cores.
2. Speed up increases with size of arrays.
3. Speed up for large arrays is much larger than speed up for small arrays.
4. Speed up per core decreases with size of cores, for most cases.
5. For small arrays and less #cores, Parallel GEMM performs slightly better than Fast GEMM.
6. For large arrays and #cores, Fast GEMM performs much better than Parallel GEMM.
Conclusions

1. Less dependencies are better.
2. Dividing into many very small tasks may not always give better results.
3. For large arrays, saving on space is also important.
Thank You