Embedded Parallel Computing

Parallel programming with Ambric

EPC Lecture 2

- Parallel programming concepts
- Lab2 Introduction
- Matrix Multiplication

Parallel thinking

- Know your application
  - Natural ways to decompose your application into tasks
  - Data access patterns
  - Data dependencies
- Be prepared to step beyond what you know
  - Requires a mix of what are traditionally hardware parallelization techniques and software techniques
  - Mix of optimizations at various levels: algorithm, data flow, instruction sequence, etc.
  - One algorithm choice may fit more naturally
- Parallelizing for Ambric is not like
  - OpenMP or POSIX Threads (Pthreads)
    - Both rely on Shared Memory Parallelism
  - Single Program Multiple Data (i.e., Vector Parallel)

Compared to Developing Hardware

- **Things which may be familiar**
  - Partitioning of tasks bounded by minimally sized memories
  - Assigning specific portions of the chip to specific tasks
- **Things which may be unfamiliar**
  - Interconnect is always 33 bits
  - All blocks have the same resources
  - No timing constraints

Compared to Developing Software

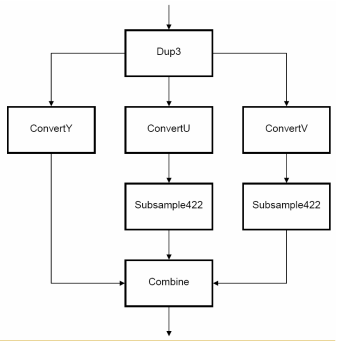
- **Things which may be familiar**
  - Analyzing your code to locate data dependencies and find available “threads”
  - Writing and optimizing your code
- **Things which may be unfamiliar**
  - You require structure (aStruct) around code objects.
  - More deterministic performance of “threads” because of encapsulation and lack of task switching
  - Smaller available memories; generally local

Parallelism - Introduction

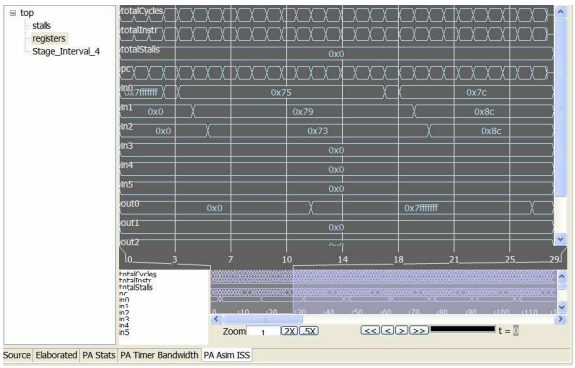
- Task Level parallelism
  - Blocks with clearly defined interfaces are good candidates.
  - Tasks may be reusable in different applications (i.e., DCT)
  - First consider your block diagram
  - Validate the I/O requirements of each block
- Data Level Parallelism
  - Select partitioning such that there are no (or limited) data dependencies between parallel elements!
  - Fanout data to multiple cores, process, and fanin
    - Fanout and fanin should use less valuable resources (SR’s)
    - Overall resource utilization should be balanced

Lab2 – Parallel Color Space Conversion

- Block diagram
  - Parallel execution
  - Pipelining



Performance Analysis

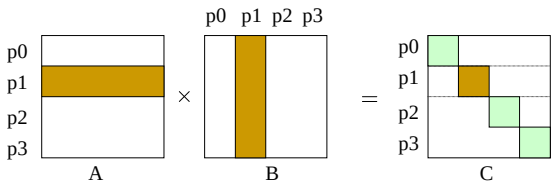


Matrix Multiplication Example

- Matrix multiplication

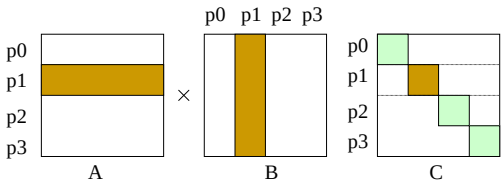
```
for (i=0; i<n; i++){
        for (j=0; j<n; j++){
                C[i][j] = 0;
                for(k=0; k<n; k++){
                        C[i][j] = C[i][j] + A[i][k]*B[k][j];
                }
        }
}
```

Matrix Multiplication On a Ring



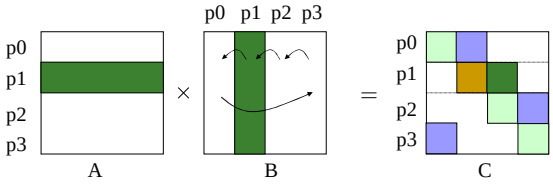
Example of matrix mapping

Matrix Multiplication Cont.



In step 1 all processors compute a part of the final matrix

Matrix Multiplication Cont.



In step 2 the sub matrix B stored in each processor are rotated left, and a new part of the final matrix is computed.

Matrix multiplication on a Mesh

- Mesh algorithm
  - assume mesh with wraparound

Step 1: Initialize matrix A by shifting row[i] i steps left (0 <= i < n)
Initialize matrix B by shifting column[j] j steps up (0 <= j < n)
Step 2: Multiply elements of A and B
Step 3: Shift matrix A one step left
Shift matrix B one step up
Step 4: Multiply elements of A and B and add to C
Step 5: repeat step 3 and step 4 n-1 times (With n rows and columns)

This also works with blocks.
Sub matrices are shifted instead of elements
(n are the number of rows and columns of processors)

Matrix multiplication on a Mesh

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]
A[3][0]	A[3][1]	A[3][2]	A[3][3]

B[0][0]	B[0][1]	B[0][2]	B[0][3]
B[1][0]	B[1][1]	B[1][2]	B[1][3]
B[2][0]	B[2][1]	B[2][2]	B[2][3]
B[3][0]	B[3][1]	B[3][2]	B[3][3]

Matrix multiplication on a Mesh

Initialization

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][2]	A[2][3]	A[2][0]	A[2][1]
A[3][3]	A[3][0]	A[3][1]	A[3][2]

B[0][0]	B[0][1]	B[0][2]	B[0][3]
B[1][0]	B[1][1]	B[1][2]	B[1][3]
B[2][0]	B[2][1]	B[2][2]	B[2][3]
B[3][0]	B[3][1]	B[3][2]	B[3][3]

Rotate A matrix left
Rotate row[i] i steps

Rotate B matrix up
Rotate col[j] j steps

Matrix multiplication on a Mesh

Initialization

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][2]	A[2][3]	A[2][0]	A[2][1]
A[3][0]	A[3][1]	A[3][2]	A[3][3]

B[0][0]	B[0][1]	B[0][2]	B[0][3]
B[1][0]	B[1][1]	B[1][2]	B[1][3]
B[2][0]	B[2][1]	B[2][2]	B[2][3]
B[3][0]	B[3][1]	B[3][2]	B[3][3]

Rotate A matrix left
Rotate row[i] i steps

Rotate B matrix up
Rotate col[j] j steps

Matrix multiplication on a Mesh

Multiplication loop

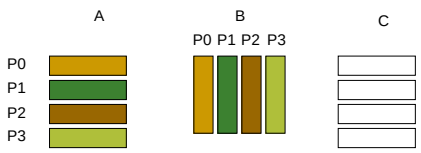
A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][2]	A[2][3]	A[2][0]	A[2][1]
A[3][0]	A[3][1]	A[3][2]	A[3][3]

B[0][0]	B[0][1]	B[0][2]	B[0][3]
B[1][0]	B[1][1]	B[1][2]	B[1][3]
B[2][0]	B[2][1]	B[2][2]	B[2][3]
B[3][0]	B[3][1]	B[3][2]	B[3][3]

Rotate A matrix left 1 step

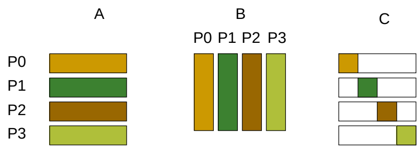
Rotate B matrix up one step

Exercise



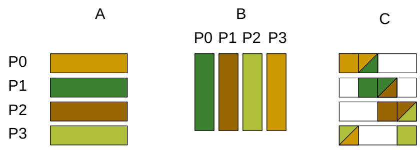
Start: Every Process contains a part of the total matrix.
Rows of A and columns of B

Exercise



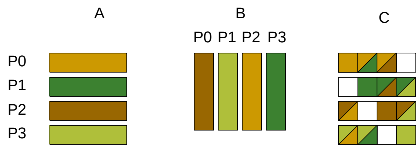
Step 1: Every process multiplies their local A and B matrices to form a part of the C matrix

Exercise



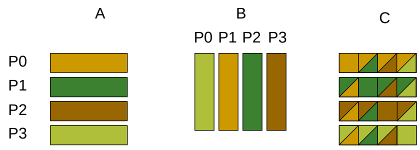
Step 2: Every process sends their B matrix one step left and multiplies the local matrices

Exercise



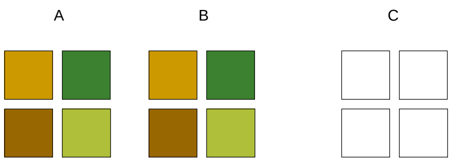
Step 2: Every process sends their B matrix one step left and multiplies the local matrices

Exercise



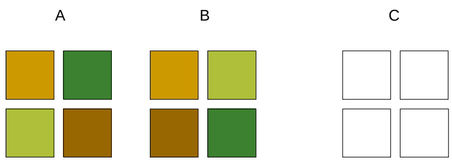
Step 2: Every process sends their B matrix one step left and multiplies the local matrices

Exercise



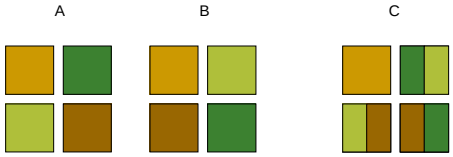
Start: Every process contains a part of the A matrix and one part of the B matrix

Exercise



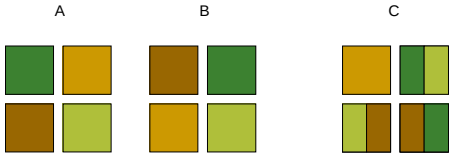
Init: Every process shifts the A matrix left as many steps as the its row coordinate.
Every process shifts the B matrix up as many steps as the its column coordinate.

Exercise



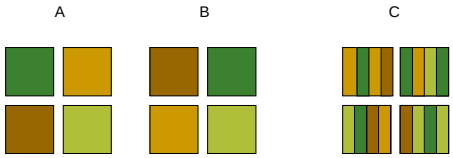
Step 1: Do a local matrix multiplication C=AB

Exercise



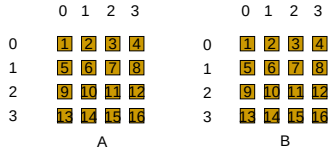
Step 2: Shift the A matrix one step left.
Shift the B matrix one step up

Exercise



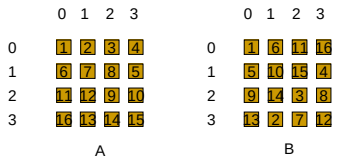
Step 3: Do a local matrix multiplication C=C+AB

Exercise



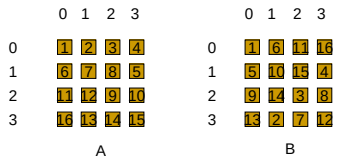
Step by step example of the matrix multiplication.
The matrix values in this example could be sub matrices instead.
Assume one value in each processor and the processors are connected in a Mesh with wraparound.

Exercise



Init: Shift A matrix left as many steps as the mesh row index.
Shift B matrix as many step up as the mesh column index.

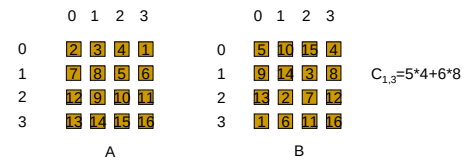
Exercise



$C_{1,3}$=5*4

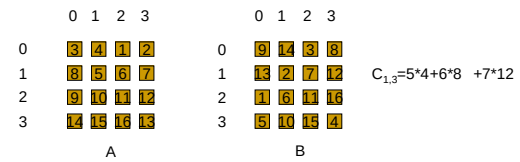
Mult: Do a local matrix multiplication. Here we look at process 1,3.

Exercise



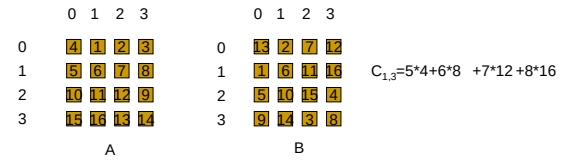
Step 1: Shift A matrix one step left and B matrix one step up.
Do a local matrix multiplication C=C+AB.

Exercise



Step 2: Shift A matrix one step left and B matrix one step up.
Do a local matrix multiplication C=C+AB.

Exercise



Step 3: Shift A matrix one step left and B matrix one step up.
Do a local matrix multiplication C=C+AB.
Done!