Message Passing Programming II

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

nCube Programming Model

See Quinn, Figure 4-12.

- SPMD Single Program, Multiple Data.
  - The same program is loaded on all nodes.
  - Every node immediately starts execution.
  - Every node operates on local data only.
  - Every node may input/output data.
- Message Passing Programming.
  - Communication by exchange of point-to-point messages.
  - Synchronization by waiting for communication partner.
  - Alternating phases of computation, communication, and waiting.

Typical for most massively parallel processors.

nCube C Extensions

- Requesting machine information.

  `whoami(&node, &d, &d, &dim)`
  Return the dimension *dim* of the allocated hypercube and the number *node* of the current processor.

- Sending a message.

  `nwrite(&msg, size, dest, type)`
  Send message at address *msg* of length *size* with identifier *type* to node *dest*. The operation does not block.

- Receiving a message.

  `nread(&msg, size, &src, &type)`
  Receive a message with identifier *type* from *src* and store it in the memory area at address *msg* of length *size*. The operation blocks until an appropriate message is available and returns the actual message length.

Receiver-side blocking.

nCube C Program

See Quinn, Figure 4-13, 4-14, 4-15.

- Numerical approximation of π.
- Local computation of subarea.
- "Fan-in" to accumulate areas.

Communication separated from program structure.

nCube Exercises

nCube/2 with 64 processors at EDVZ.

- Login.

  `telnet ncube.edvz.uni-linz.ac.at`

- Manual pages.

  `nman ncube`

- Compiling.

  `ncc -O prog.c -o prog`

- Machine status.

  `nwho`

- Executing.

  `xnc -d`*`cubedim`*` prog`

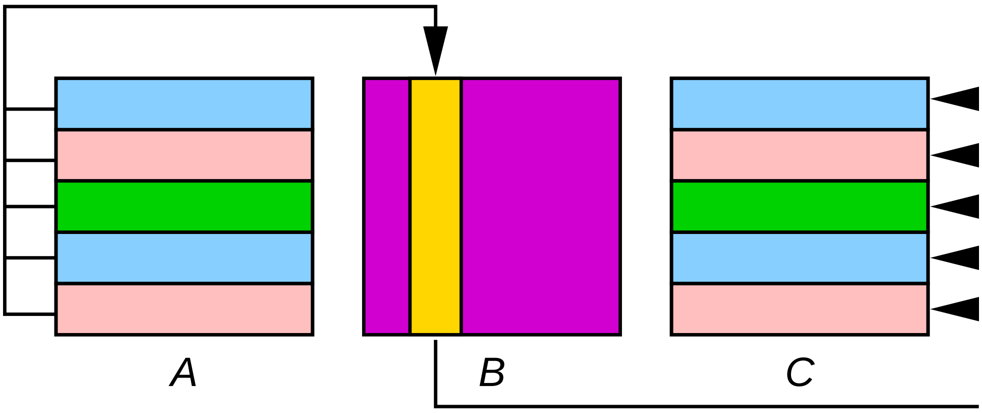
- Timing.

  `clock()`, `micclk()`

Manual pages!

Matrix Multiplication

Row-Column Oriented Algorithm



1. Load A_i on every processor P_i.
2. For all P_i do:

```
for j=0 to N-1
  Receive B_j from root
  C_ij = A_i * B_j
```

3. Collect C_i

Broadcasting of each $B_j \Rightarrow$ *Step 2 takes* $O(N \log N)$ *time.*

Ring Algorithm

See Quinn, Figure 7-15.

- Change order of multiplication by
- Using a *ring* of processors.

1. Load A_i and B_i on every processor P_i.
2. For all P_i do:

```
p = (i+1) mod N
j = i
for k=0 to N-1 do
   C_ij = A_i * B_j
   j = (j+1) mod N
   Receive B_j from P_p
```

3. Collect C_i

Point-to-point communication $\Rightarrow$ *Step 2 takes* $O(N)$ *time.*

Hypercube Algorithm

Problem: How to embed ring into hypercube?

- Simple solution $H(i) = i$:
  - Ring processor i is mapped to hypercube processor $H(i)$.
  - Massive non-neighbor communication!
- How to preserve neighbor-to-neighbor communication? (see Quinn, Figure 5-13)
- Requirements for $H(i)$:
  - H must be a 1-to-1 mapping.
  - $H(i)$ and $H(i+1)$ must differ in 1 bit.
  - $H(0)$ and $H(N-1)$ must differ in 1 bit.

Can we construct such a function H?

Ring Successor

Assume H is given.

- Given: hypercube processor number i
- Wanted: “ring successor” $S(i)$

$$S(i) = \begin{cases} 0, & \text{if } i = N-1 \\ H(H^{-1}(i)+1), & \text{otherwise} \end{cases}$$

Same technique for embedding a 2-D mesh into an hypercube (see Quinn, Figure 5-14).

Gray Codes

Recursive construction.

- 1-bit Gray code G_1

i	$G_1(i)$
0	0
1	1

- n-bit Gray code G_n

i	$G_n(i)$	i	$G_n(i)$
0	$0G_{n-1}(0)$	$n-1$	$1G_{n-1}(0)$
1	$0G_{n-1}(1)$	$n-2$	$1G_{n-1}(1)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$\frac{n}{2}-1$	$0G_{n-1}(\frac{n}{2}-1)$	$\frac{n}{2}$	$1G_{n-1}(\frac{n}{2}-1)$

- Required properties preserved by construction!

$$H(i) = G(i) = i \text{ xor } \frac{i}{2}.$$

Gray Code Computation

C functions.

- Gray-Code

```
int G(int i)
{
  return(i ^ (i/2));
}
```

- Inverse Gray-Code

```
int G_inv(int i)
{
  int answer, mask;
  answer = i;
  mask = answer/2;
  while (mask > 0)
  {
    answer = answer ^ mask;
    mask = mask / 2;
  }
  return(answer);
}
```

Block-Oriented Algorithm

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} =$$

$$\begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

- Use block-oriented distribution introduced for shared memory multiprocessors.

  Block-matrix multiplication is analogous to scalar matrix multiplication.

- Use staggering technique introduced for 2D SIMD mesh.

  Rotation along rows and columns.

- Perform the SIMD matrix multiplication algorithm on whole *submatrices*.

  Submatrices are multiplied and shifted.

Analysis of Algorithm

n^2 matrix, p processors.

- Row-Column-oriented algorithm
  - Computation: $n^2/p * n/p = n^3/p^2$.
  - Communication: $2(\lambda + \beta n^2/p)$
  - p iterations.
- Block-oriented algorithm
  - Computation: $n^2/p * n/p = n^3/p^2$.
  - Communication: $4(\lambda + \beta n^2/p)$
  - $\sqrt{p} - 1$ iterations.
- Comparison

$$2p(\lambda + \beta n^2/p) > 4(\sqrt{p} - 1)(\lambda + \beta n^2/p)$$
$$\lambda p + \beta n^2 > 4\lambda(\sqrt{p} - 1) + 4\beta(\sqrt{p} - 1)n^2/p$$

1. $p > 4(\sqrt{p} - 1) \;\rightarrow\; p > 4$
2. $1 > 4(\sqrt{p} - 1)/p \;\rightarrow\; p > 4$

For larger processor numbers, communication dominates computation!