Parallel Inner Product Computation

Section 1.3 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

Utrecht University

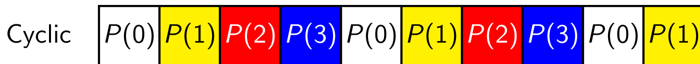
Inner product of two vectors

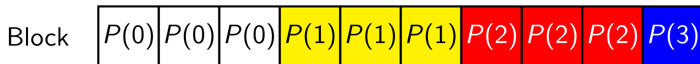
The inner product of two vectors $\mathbf{x} = (x_0, \ldots, x_{n-1})^{\mathrm{T}}$ and $\mathbf{y} = (y_0, \ldots, y_{n-1})^{\mathrm{T}}$ is defined by

$$\alpha = \mathbf{x}^{\mathrm{T}}\mathbf{y} = \sum_{i=0}^{n-1} x_i y_i.$$

Here, 'T' denotes transposition. All vectors are column vectors.

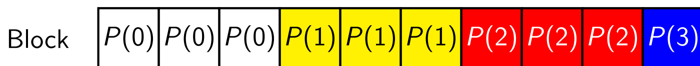
Data distributions for a vector

Cyclic


Block


$p = \#$ processors $= 4$
$n =$ vector length $= 10$

Block distribution



- The block distribution is defined by

$$x_i \longmapsto P(i \text{ div } b), \text{ for } 0 \le i < n.$$

- Here, the div operator stands for dividing and rounding down:

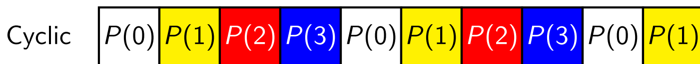
$$i \text{ div } b = \lfloor i/b \rfloor.$$

- The block size is $b = \lceil \frac{n}{p} \rceil$ (rounded up).

Load balance

- For $n = 9$ and $p = 4$, we have a block size $b = \lceil \frac{9}{4} \rceil = 3$, so the block distribution assigns 3, 3, 3, 0 vector components to the processors, respectively.
- The load balance of an algorithm is determined by the processor with the maximum amount of work, here 3.
- For good load balance, this should be close to the average amount of work, here 2.25.
- A variant of the block distribution would assign 3, 2, 2, 2 components. Just as good!

Cyclic distribution



- The cyclic distribution is defined by

$$x_i \longmapsto P(i \bmod p), \text{ for } 0 \leq i < n.$$

- This distribution is based on the modulo-operator.
- For $p = 4$, x_7 is assigned to processor $P(7 \bmod 4) = P(3)$.
- Starting to count at 0 simplifies the formula.

Some kids have been raised
to start counting at 0.
Now they work in C.

Parallel inner product computation

Design decisions:

- Assign x_i and y_i to the same processor, for all i. This makes computing $x_i \cdot y_i$ a local operation. Thus $\mathrm{distr}(\mathbf{x}) = \mathrm{distr}(\mathbf{y})$.
- Choose a distribution with an even spread of vector components. Both block and cyclic distributions are fine. We choose cyclic, following the way card players deal their cards.
- The data distribution naturally leads to a work distribution and a parallel algorithm.

Example for $n = 10$ and $p = 4$

$\mathbf{x}$	12	0	4	7	-1	2	15	11	3	-2
$\cdot$										
$\mathbf{y}$	1	9	2	0	-1	12	1	1	3	8
$+$										
	22					8				

23 11

22 8 23 11 | 22 8 23 11 | 22 8 23 11 | 22 8 23 11

$+$

α 64 64 64 64

Parallel inner product algorithm for $P(s)$

input: $\mathbf{x}, \mathbf{y}$: vector of length n,
$\mathrm{distr}(\mathbf{x}) = \mathrm{distr}(\mathbf{y}) = \phi$,
$\phi(i) = i \bmod p$, for $0 \leq i < n$.

output: $\alpha = \mathbf{x}^{\mathrm{T}}\mathbf{y}$, $\mathrm{repl}(\alpha) = P(*)$.

$\alpha_s := 0$; ▷ Superstep (0)
for $i := s$ **to** $n-1$ **step** p **do**
$\alpha_s := \alpha_s + x_i y_i$;

Parallel inner product algorithm for $P(s)$

input: $\mathbf{x}, \mathbf{y}$: vector of length n,
$\mathrm{distr}(\mathbf{x}) = \mathrm{distr}(\mathbf{y}) = \phi$,
$\phi(i) = i \bmod p$, for $0 \le i < n$.

output: $\alpha = \mathbf{x}^{\mathrm{T}}\mathbf{y}$, $\mathrm{repl}(\alpha) = P(*)$.

$\alpha_s := 0;$ ▷ Superstep (0)
for $i := s$ **to** $n - 1$ **step** p **do**
$\alpha_s := \alpha_s + x_i y_i;$

for $t := 0$ **to** $p - 1$ **do** ▷ Superstep (1)
put α_s in $P(t)$;

Parallel inner product algorithm for $P(s)$

input: $\mathbf{x}, \mathbf{y}$: vector of length n,
$\mathrm{distr}(\mathbf{x}) = \mathrm{distr}(\mathbf{y}) = \phi$,
$\phi(i) = i \bmod p$, for $0 \leq i < n$.

output: $\alpha = \mathbf{x}^{\mathrm{T}}\mathbf{y}$, $\mathrm{repl}(\alpha) = P(*)$.

$\alpha_s := 0$; ▷ Superstep (0)
for $i := s$ **to** $n-1$ **step** p **do**
  $\alpha_s := \alpha_s + x_i y_i$;

for $t := 0$ **to** $p-1$ **do** ▷ Superstep (1)
  put α_s in $P(t)$;

$\alpha := 0$; ▷ Superstep (2)
for $t := 0$ **to** $p-1$ **do**
  $\alpha := \alpha + \alpha_t$;

Single Program, Multiple Data (SPMD)

- Only one program text needs to be written. All processors run the same program, but on their own data.
- The program text is parametrized in the processor number s, $0 \leq s < p$, also called processor identity. The actual execution of the program depends on s.
- Processor $P(s)$ computes a local partial inner product

$$\alpha_s = \sum_{\substack{0 \leq i < n \\ i \bmod p = s}} x_i y_i.$$

- The corresponding computation superstep (0) has 1 addition and 1 multiplication per local vector component and costs

$$2 \left\lceil \frac{n}{p} \right\rceil + l.$$

Redundant computation

- The partial inner products must be added. This could have been done by $P(0)$, i.e., processor 0.
- Sending the α_s to $P(0)$ is a $(p-1)$-relation. Sending them to $P(*)$, i.e., to all the processors, costs the same. The cost is $(p-1)g + l$.
- Computing α on $P(0)$ costs the same as computing it on all the processors redundantly, i.e., in a replicated fashion. The cost is $p + l$.
- Therefore, we send α_s to all the processors and compute α redundantly.
- This approach saves the superstep of sending α back from $P(0)$ to all other processors, which would cost $(p-1)g + l$.
- The reduction from the local α_s to a single number α available on all processors is sometimes called an Allreduce operation.

Result needed on all processors

- Often, the result is needed on all processors. An example is iterative linear system solvers. The algorithm does just this.
- Sending the local result to all processors is best if each processor contributes one value.
- If there are more values per processor, then a different approach might be better.

Total BSP cost of inner product algorithm

$$T_{\mathrm{inprod}} = 2\left\lceil \frac{n}{p} \right\rceil + p + (p-1)g + 3l.$$

One-sided communication

- We expressed communication by using the '**put**' operation, which involves an active sender and a passive receiver.
- We assume all puts are accepted. Thus, we can define each data transfer by giving only the action of one side.
- No clutter in programs: shorter and simpler texts.
- No danger of the dreaded deadlock. What happens if both processors want to receive first?
- Deadlock can easily occur in message passing, with an active sender and an active receiver that must shake hands, or kiss. This may cause lots of problems.
- Another one-sided communication is the 'get'.
- One-sided communications are more efficient.

Summary

- We design algorithms in Single Program, Multiple Data style. Each processor runs its own copy of the same program, on its own data.
- The block and cyclic distributions are commonly used in parallel computing. Both are suitable for an inner product computation.
- The BSP style encourages balancing the communication among the processors. Sending all data to one processor is discouraged. Better: all to all.
- One-sided communications such as puts and gets are easy to use and efficient.