CME 213

SPRING 2012-2013

Eric Darve

Stanford University

Performance metrics

THE BASIC CONCEPT: SPEED-UP

- This quantity measures how much faster the code runs because we are using many processes.
- Define:
  $T^\star(n)$
  the optimal (reference) running time with a single process.
- Define:
  $T_p(n)$
  the running time with p processes.
- The speed-up is then the ratio:

$$S_p(n) = \frac{T^\star(n)}{T_p(n)}$$

- We expect this number to go up as p as we keep increasing the number of processes.

A MORE APPROPRIATE CONCEPT: EFFICIENCY

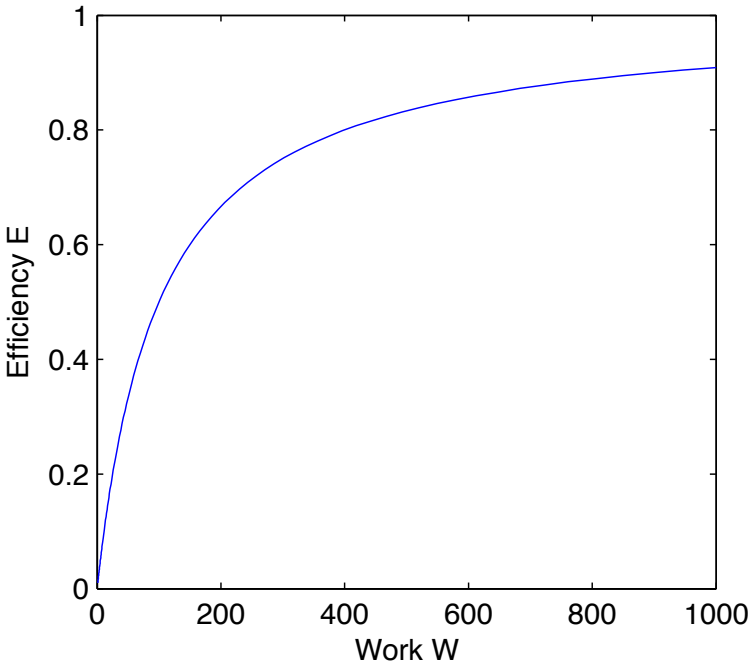
- The previous definition has a problem. As p increases, we want to know whether the speed-up scales as p or not.
- This might be difficult to assess from a plot. Ideally the speed-up is a straight line.
- It is therefore more convenient to look at the efficiency:

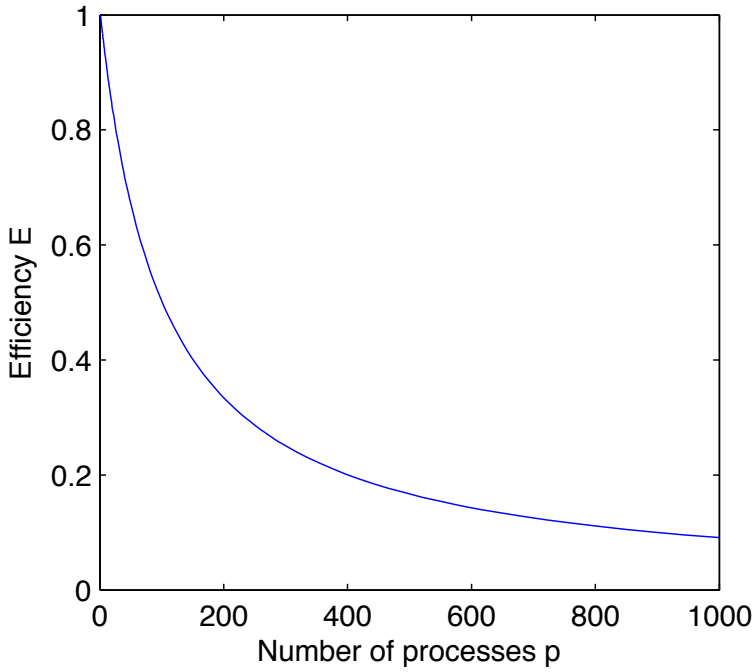
$$E_p(n) = \frac{S_p(n)}{p} = \frac{T^\star(n)}{pT_p(n)}$$

- Ideally that quantity is simply a constant as p increases. That is easier to read from a plot.
- The maximum value for efficiency is 1 (except in some rare circumstances because of cache effects).

EFFICIENCY PLOTS

Typical behavior of the efficiency as the number of processes increases or as the problem size (amount of computational work to perform) increases.





EXAMPLE 1: DOT PRODUCT

Two-step algorithm:

1. Calculate local dot product:

$$\sum_j a_j \; b_j$$

2. Use a spanning tree for the final reduction: $\ln_2$ p passes are required.

Total run time with one process:

$$T^\star(n) = \alpha n$$

Total run time in parallel:

$$T_p(n) = \alpha n/p + \beta \ln_2 p$$

Efficiency

- Efficiency:

$$E_p(n) = \frac{\alpha n}{\alpha n + \beta\, p \log_2 p} = \frac{1}{1 + (\beta/\alpha)\,(p \log_2 p)/n}$$

- The efficiency can be maintained (iso-efficiency) provided that we do not scale p faster than:

$$p \log_2 p = \Theta(n) \quad \text{or} \quad p = \Theta(n/\log_2 n)$$

EXAMPLE 2: MATRIX-VECTOR PRODUCT WITH 1D PARTITIONING

- In that case we can model the serial running time as:

$$T^{\star}(n) = \alpha n^2$$

- Parallel running time:

$$T_p(n) = \alpha n^2/p + \beta \ln p + \gamma n$$

- Efficiency:

$$E_p(n) = \frac{1}{1 + (\beta/\alpha)(p \ln p)/n^2 + (\gamma/\alpha)p/n}$$

- Iso-efficiency requires:

$$p = \Theta(n)$$

EXAMPLE 2: MATRIX-VECTOR PRODUCT WITH 2D PARTITIONING

- Computation:

$$\alpha n^2/p$$

- Send b to diagonal processes:

$$\beta + \gamma\, n/\sqrt{p}$$

- Broadcast b in each column:

$$(\beta + \gamma\, n/\sqrt{p}) \ln \sqrt{p}$$

- Reduction across columns (same running time as broadcast because these operations are dual of one another)

$$(\beta + \gamma\, n/\sqrt{p}) \ln \sqrt{p}$$

ISO-EFFICIENCY

- With the previous results:

$$E_p(n) = \frac{1}{1 + (\beta/\alpha)(p \ln p)/n^2 + (\gamma/\alpha)(p^{1/2} \ln p)/n}$$

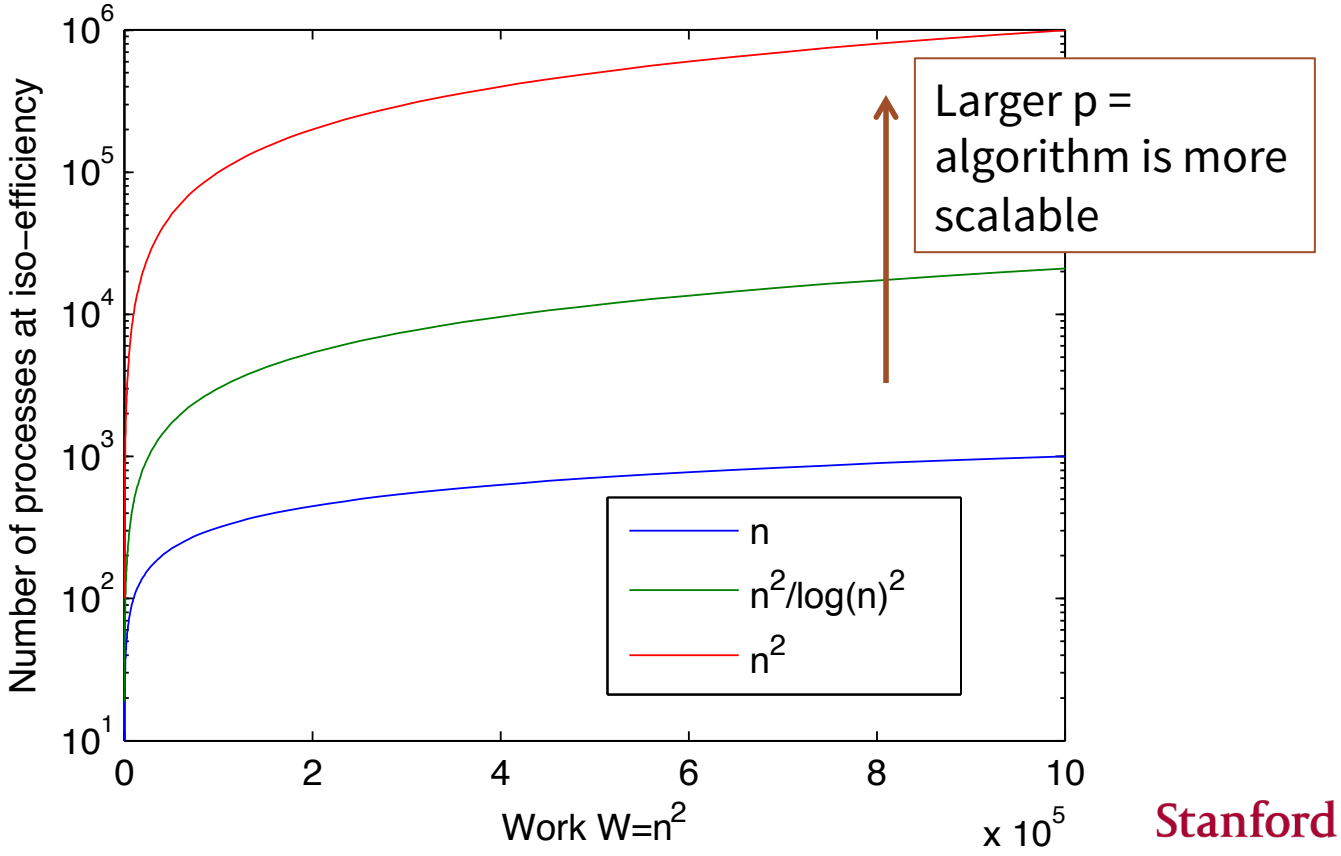
- Iso-efficiency:

$$p = \Theta\left(n^2/(\ln n)^2\right) \gg \Theta(n)$$

- This is much better than with the 1D partitioning. We can increase p much faster at iso-efficiency. The scalability is much improved.

ISO-EFFICIENCY PLOTS

- We plot for the two algorithms the value of p as a function of n such that iso-efficiency is maintained.
- Larger values of p are better.
- This means improved scalability.



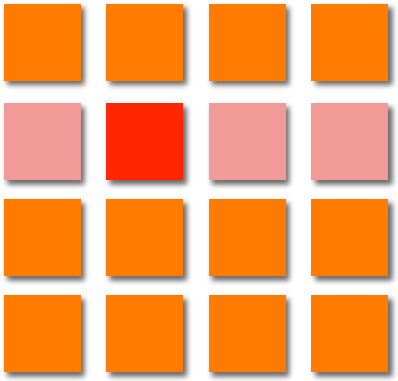
MATRIX-MATRIX PRODUCTS

Algorithm in pseudo-code:

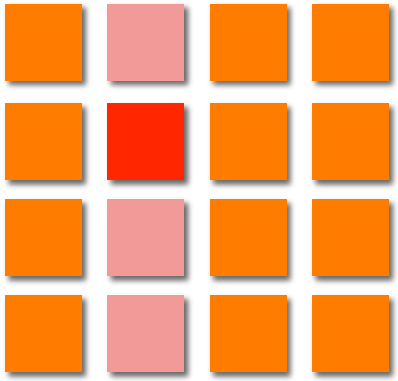
```
for i=0:n-1 do
    for j=0:n-1 do
        C(i,j) = 0;
        for k=0:n-1
            C(i,j) += A(i,k)*B(k,j);
        end
    end
end
```

BLOCK OPERATIONS

- Algorithm proceeds by doing block operations.



Matrix A



Matrix B

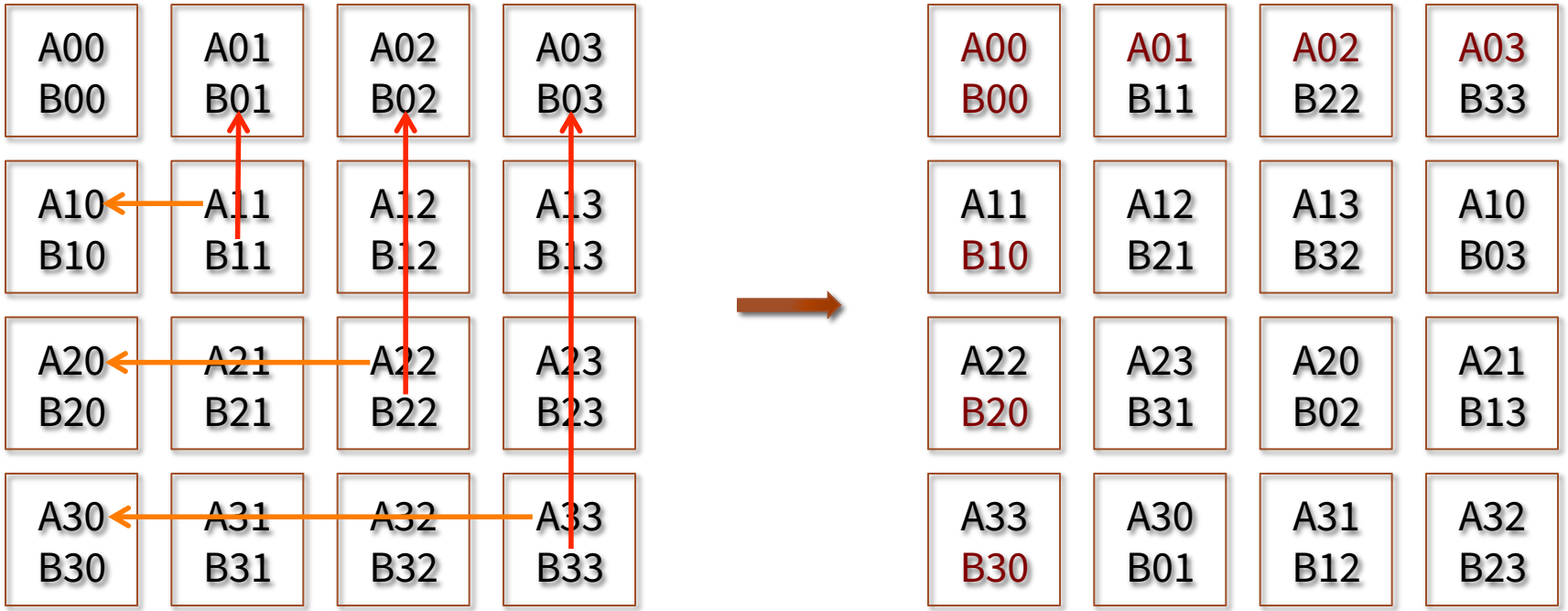
- For p processes, we create p blocks of size $n/p^{1/2}$.
- Simple approach:
  - all-to-all broadcast in each row of A
  - all-to-all broadcast in each column of B
  - Perform calculation with local data on each process
- Iso-efficiency:

$$p = \Theta(n^2)$$

CANNON'S ALGORITHM

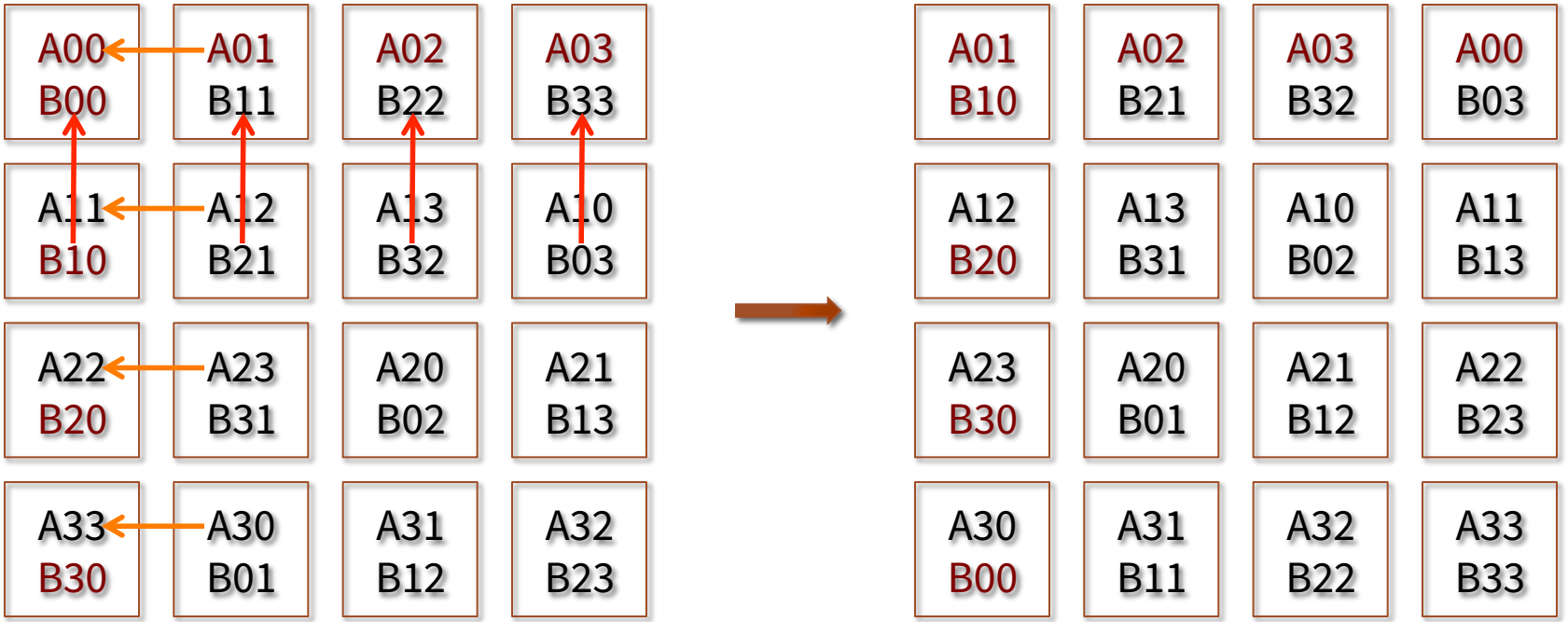
- There are two issues with this simple algorithm:
  - We should be able to increase p closer to n^3 at iso-efficiency.
  - This algorithm requires a lot of memory since a process needs to store an entire block row of A and block column of B.
- Cannon's algorithm allows reducing the memory footprint.
- It works by cleverly shuffling the blocks of A and B such that each process never stores more than one block of A and B.
- The blocks of A are rotated inside each row while the blocks of B are rotated inside each column.
- The trick is to start with the right alignment.

COMMUNICATION STEPS



- After the first communication step, each process has data to perform its first block multiplication.
- A key point is that, from then on, only a simple communication with neighbors is required at each step.

First shift



- B blocks are shifted up while A blocks are shifted left.
- Each process has now the next two blocks required for the product.
- A similar second and third shifts are required to complete the calculation: shift B up and A left.

CANNON'S ALGORITHM

- With this algorithm, processes store only 2 blocks at a time.
- Cost of communication is slightly different from naïve algorithm but in the end the running times are comparable.
- The iso-efficiency curve is very close to the other algorithm with p scaling as n^2.

Dekel-Nassimi-Sahni algorithm

- The number of operations is $O(n^3)$. Therefore we should be able to use up to $p = O(n^3)$ processes.
- In this algorithm, each process P_{ijk} calculates one product A(i,k)B(k,j) (where these are blocks in general).
- When the algorithm starts, only a subset of the processes are needed to store A and B. $O(p^{2/3})$ are used.
- Step 1: communicate A(i,k) and B(k,j) to process P_{ijk}.
- Step 2: calculate product of two blocks.
- Step 3: perform a reduction over k so that P_{ij0} has the resulting block C(i,j).

- The iso-efficiency curve for this algorithm is:

$$p = \Theta\left(n^3/(\ln n)^3\right)$$

WHY IS DNS A BETTER ALGORITHM?

- What does it mean that the iso-efficiency curve of DNS is better than Cannon?
- In Cannon, $p = O(n^2)$, whereas in DNS $p = O(n^3/\ln^3 n)$.
- Does that mean DNS need more processes than Cannon? No. It means DNS remains efficient even with larger p.
- Assume now that p is fixed. Because of the difference in the two algorithms, in Cannon, the block size $n/p^{1/2}$ is smaller than the block size $n/p^{1/3}$ in DNS. Recall that $p^{1/2} > p^{1/3}$.
- Because DNS is able to use larger blocks, fewer communications are required. The algorithm is therefore more efficient.
- This translates directly into a better iso-efficiency curve:

$$n^3/(\ln n)^3 \gg n^2$$

Conclusion and summary

Two main parallel programming paradigms:

1. Shared memory:
    1. Multicore processors: Pthreads, openMP
    2. Graphics processors: CUDA
2. Distributed memory: MPI

Language	Processor type	Grain size	Threads	Data commu-nication	Programming
Pthread **OpenMP**	Multicore	**Mid**-size grained	**Few**; all are live.	Read and write to **shared** memory.	**Pthreads**: completely explicit and very flexible. **OpenMP**: only few lines of code need to be added. Not always easy to understand what the compiler actually does. More restrictive.
CUDA	GPU	**Fine** grained	**Large number** of threads created; a subset is resident at any given time.	Read and write to **shared** memory.	Initial implementation is relatively easy. Advanced optimization usually required.
MPI	Computer node	**Coarse** grained: bulk computa-tion	**One process per core.** Number of cores can be increased “as needed.”	Requires **message passing** across a network.	Conceptually simple but many lines of code often required. Large library of functions.