Comp 336 & 529
Big Data Analytics

Science and Technology Facilities Council

Hartree Centre

INTELLECTUAL PROPERTY RIGHTS NOTICE:

For further details, please email us: hartreetraining@stfc.a.c.uk

Science and Technology Facilities Council

Hartree Centre

INTELLECTUAL PROPERTY RIGHTS NOTICE:

There is some material from A.Grama, A. Gupta, G. Karypis, V.Kumar ``Introduction to Parallel Computing'', and Jim Demmel, University of Berkeley, USA, used in this lecture.

Please note that the corresponding Intellectual Property Rights apply as per University of Berkeley IPR rules for any material from Jim Demmel's lectures and Addison Wesley's IPR rules apply for the material from "Introduction to Parallel Computing" book.

Science and Technology Facilities Council

Hartree Centre

Week 8: Scalable algorithms for analysing large datasets – Part 1

Prof Vassil Alexandrov | Dr. Dominic Richards

Hartree Centre

Parallel Computing – key concepts and performance metrics

Overview

- Sources of Overhead in Parallel Programs
- Performance Metrics for Parallel Systems
- Scalability of Parallel Algorithms
- Asymptotic Analysis of Parallel Algorithms
- Energy Efficiency and communication avoiding algorithms

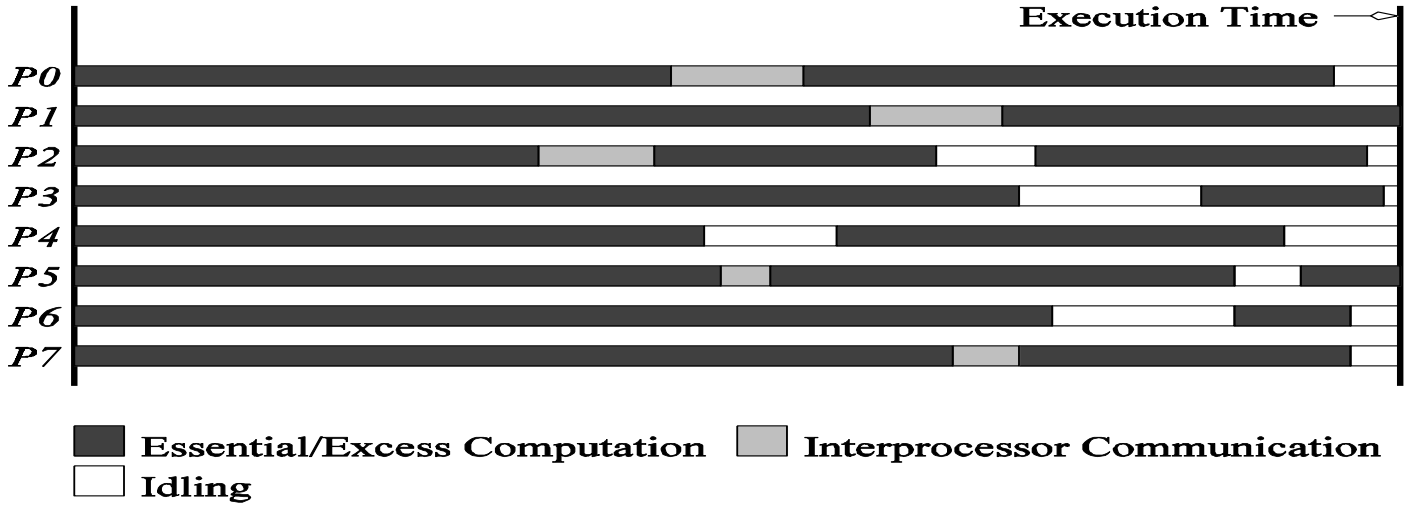
Some material from A.Grama, A. Gupta, G. Karypis, V.Kumar
``Introduction to Parallel Computing'', Addison Wesley, 2003.

Parameters

- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).
- The parallel runtime of a program depends on several parameters: on the input size, the number of processors, and the communication parameters of the machine.
- Wall clock time - the time from the start of the first processor to the stopping time of the last processor in a parallel ensemble.
- How much faster is the parallel version? What is the baseline serial version with which we compare? Can we use a suboptimal serial program to make our parallel program look
- Raw FLOP count - What good are FLOP counts when they do not solve a problem?

Sources of Overhead in Parallel Programs

- Idling
- Communication
- Synchronization
- Excessive computations



Performance Metrics for Parallel Algorithms

- Serial runtime of a program is the time elapsed between the beginning and the end of its execution on a sequential computer T_S .
- The parallel runtime is the time that elapses from the moment the first processor starts to the moment the last processor finishes execution.
- We denote the serial runtime by T_S and the parallel runtime by T_P.

Performance Metrics for Parallel Algorithms: Total Parallel Overhead

- Let T_{all} be the total time collectively spent by all the processing elements.
- T_S is the serial time.
- Observe that $T_{all} - T_S$ is then the total time spend by all processors combined in non-useful work. This is called the *total overhead*.
- The total time collectively spent by all the processing elements $T_{all} = p\ T_P$ (p is the number of processors).
- The overhead function (T_o) is therefore given by

$$T_o = p\ T_P - T_S \qquad (1)$$

Performance Metrics for Parallel Algorithms: Speedup

- Speedup (***S***) is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with ***p*** identical processing elements.

- Speedup $Sp=Ts/Tp$

- Speedup, in theory, should be upper bounded by ***p*** - after all, we can only expect a ***p***-fold speedup if we use times as many resources.
- Usually $0< Sp < p$

Performance Metrics: Speedup

- For a given problem, there might be many serial algorithms available. These algorithms may have different asymptotic runtimes and may be parallelizable to different degrees.
- For the purpose of computing speedup, we always consider the best sequential program as the baseline.
- Consider a parallel bubble sort with a serial time of 160 seconds.
- The parallel time for odd-even sort (efficient parallelization of bubble sort) is 40 seconds.
- The speedup would appear to be 160/40 = 4.
- What if the serial quicksort only took 20 seconds? In this case, the speedup is 20/40 = 0.5. This is a more realistic assessment.

Performance Metrics: Efficiency

- Efficiency is a measure of the fraction of time for which a processing element is usefully employed

- Mathematically, it is given by

$$E = \frac{S}{p}. \qquad (2)$$

$$0 < E < 1$$

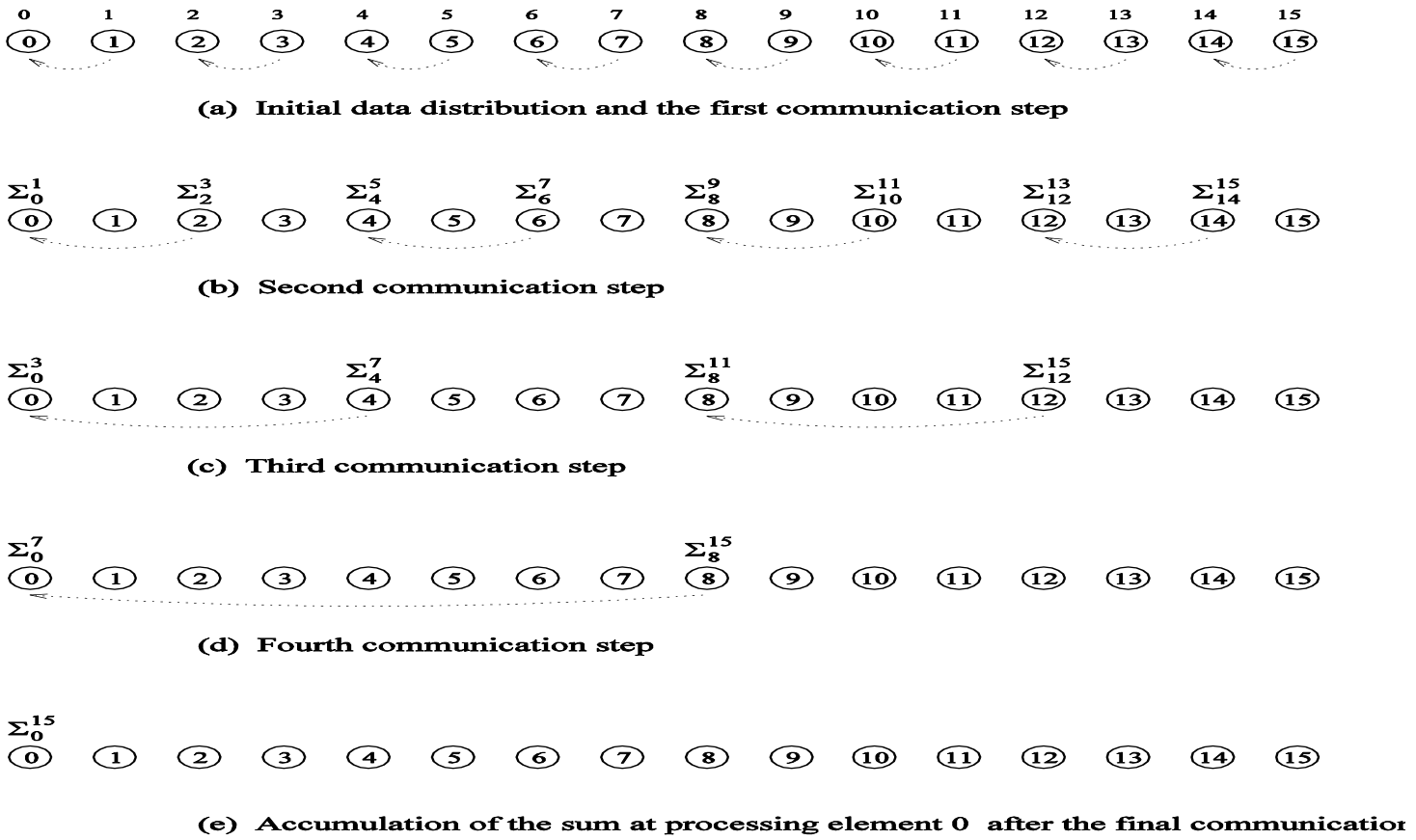
- Following the bounds on speedup, efficiency can be as low as 0 and as high as 1.

Performance Metrics: Example

- Consider the problem of adding ***n*** numbers by using ***n*** processing elements.

- If ***n*** is a power of two, we can perform this operation in **log *n*** steps by propagating partial sums up a logical binary tree of processors.

Performance Metrics: Example



Computing the global sum of 16 partial sums using 16 processing elements .
Σ^{j}_{i} denotes the sum of numbers with consecutive labels from ***i*** to ***j***.

Performance Metrics: Example (continued)

- If an addition takes constant time, say, t_c and communication of a single word takes time $t_s + t_w$, we have the parallel time

$$T_P = \Theta (\log n)$$

- We know that $T_S = \Theta (n)$
- Speedup S is given by $S = \Theta (n / \log n)$

Performance Metrics: Efficiency Example

- The speedup of adding numbers on processors is given by

$$S = \frac{n}{\log n}$$

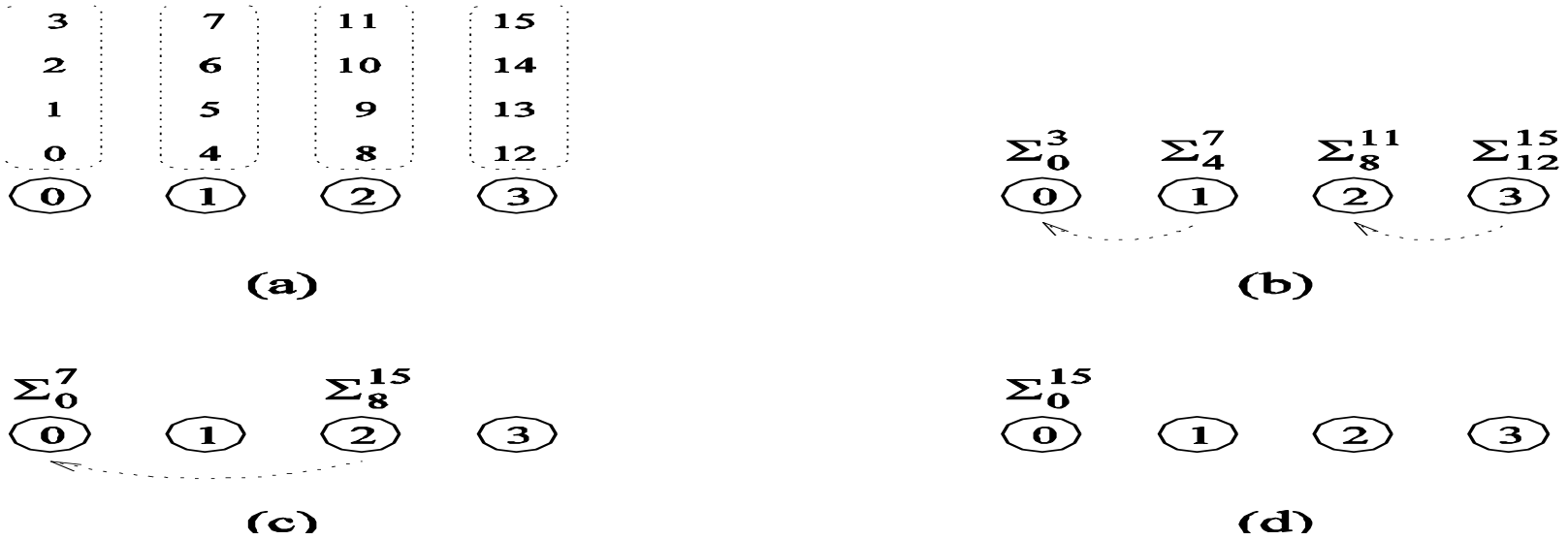
- Efficiency is given by

$$E = \frac{\Theta\left(\frac{n}{\log n}\right)}{n}$$

$$\Theta\left(\frac{1}{\log n}\right)$$

Can we do better: Example (continued)

Each processing element locally adds its n / p numbers in time $\Theta(n / p)$.
The p partial sums on p processing elements can be added in time $\Theta(n / p)$.



Scaling Characteristics of Parallel Algorithms: Example

For the problem of adding n numbers on p processing elements we have:

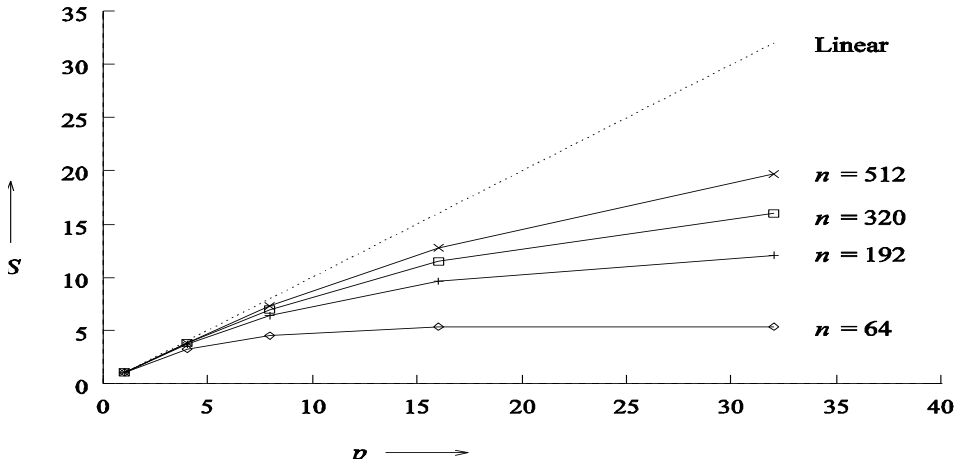
$$T_P = \frac{n}{p} + 2\log p \tag{5}$$

$$S = \frac{n}{\frac{n}{p} + 2\log p} \tag{6}$$

$$E = \frac{1}{1 + \frac{2p\log p}{n}} \tag{7}$$

Parallel Algorithms and scalability: Example (cont.)

Plotting the speedup for various input sizes:



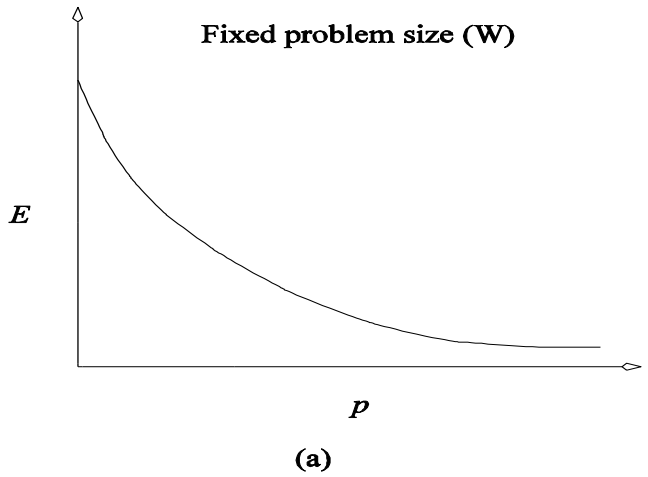
Speedup versus the number of processing elements for adding a list of numbers.

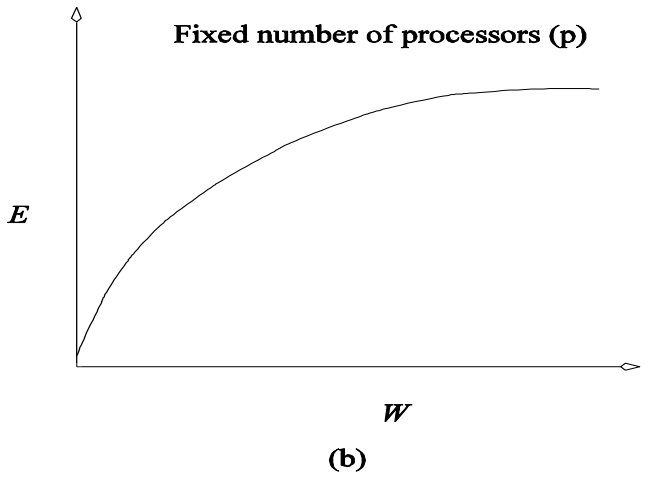
Total overhead function T_o is a function of both problem size T_s and the number of processing elements p.

In cases when T_o grows sublinearly with respect to T_s. the efficiency increases if the problem size is increased keeping the number of processing elements constant

In such cases we can simultaneously increase the problem size and number of processors to keep efficiency constant.

Isoefficiency Metric of Scalability





Variation of efficiency:

(a) as the number of processing elements is increased for a given problem size;

(b) as the problem size is increased for a given number of processing elements.

(c) The phenomenon illustrated in graph (b) is not common to all parallel programs.

Isoefficiency Metric of Scalability

- What is the rate at which the problem size must increase with respect to the number of processing elements to keep the efficiency fixed?
- This rate determines the scalability of the system. The slower this rate, the better.
- Before we formalize this rate, we define the problem size ***W*** as the asymptotic number of operations associated with the best serial algorithm to solve the problem.

Isoefficiency Metric of Scalability

- We can write parallel runtime as:

$$T_P = \frac{W + T_o(W,p)}{p} \tag{8}$$

- The resulting expression for speedup is

$$\begin{aligned} S &= \frac{W}{T_P} \\ &= \frac{Wp}{W + T_o(W,p)}. \end{aligned} \tag{9}$$

- Finally, we write the expression for efficiency as

$$\begin{aligned} E &= \frac{S}{p} \\ &= \frac{W}{W + T_o(W,p)} \\ &= \frac{1}{1 + T_o(W,p)/W}. \end{aligned}$$

Isoefficiency Metric of Scalability

- For scalable parallel program, efficiency can be maintained at a fixed value (between 0 and 1) if the ratio T_o / W is maintained at a constant value.
- For a desired value E of efficiency,

$$
\begin{aligned}
E &= \frac{1}{1 + T_o(W,p)/W}, \\
\frac{T_o(W,p)}{W} &= \frac{1-E}{E}, \\
W &= \frac{E}{1-E} T_o(W,p).
\end{aligned}
\tag{11}
$$

- If $K = E / (1 - E)$ is a constant depending on the efficiency to be maintained, since T_o is a function of W and p, we have

$$
W = K T_o(W,p). \tag{12}
$$

Isoefficiency Metric: Example

- The overhead function for the problem of adding ***n*** numbers on ***p*** processing elements is approximately ***2p* log *p*** .

- Substituting T_o by ***2p* log *p*** , we get

$$W = K2p \log p. \quad (13)$$

Thus, the asymptotic isoefficiency function for this parallel system is

$$\Theta(p \log p)$$

- If the number of processing elements is increased from ***p*** to ***p'***, the problem size (in this case, n) must be increased by a factor of (***p'* log *p'***) / (***p* log *p***) to get the same efficiency as on ***p*** processing elements.

Amdahl's Law: Fixed problem size

In many practical applications the computational workload is fixed:

Two parts for the problem with size W : sequential and parallel part

$W = \alpha W + (1- \alpha)W$

$Sp = W / (\alpha W + (1- \alpha)(W/p))$

$= p / (1 + (p - 1) \alpha) \rightarrow 1 / \alpha$ as $p \rightarrow \infty$

Speedup is limited by $1 / \alpha$

Science and Technology Facilities Council

Hartree Centre

Communication-Avoiding Algorithms and Scalability

Some material from Prof. Jim Demmel's lectures, EECS & Math Departments, UC Berkeley

www.cs.berkeley.edu/~demmel/SC12_tutorial

Science and Technology Facilities Council

Hartree Centre

Examples:

- Matrix multiply
  - Lower bounds on communication
  - New algorithms that attain these lower bounds

- Similar approaches work for
  "Direct" Linear Algebra

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Lower bound for all “direct” linear algebra

- Let M = “fast” memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul,

Lower bound for all “direct” linear algebra

- Let M = “fast” memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent ≥ #words_moved / largest_message_size

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size (per processor)

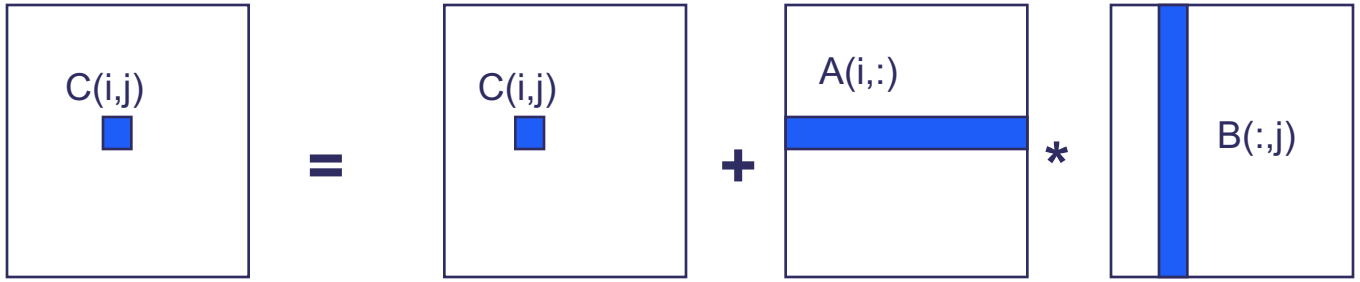
#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent (per processor) = Ω(#flops (per processor) / $M^{3/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

Naïve Matrix Multiply

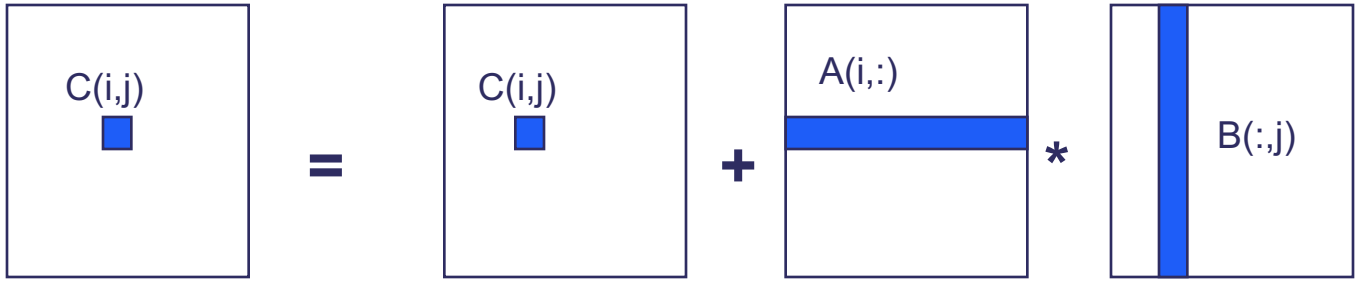
```
{implements C = C + A*B}
for i = 1 to n

  for j = 1 to n



    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
```



Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Naïve Matrix Multiply

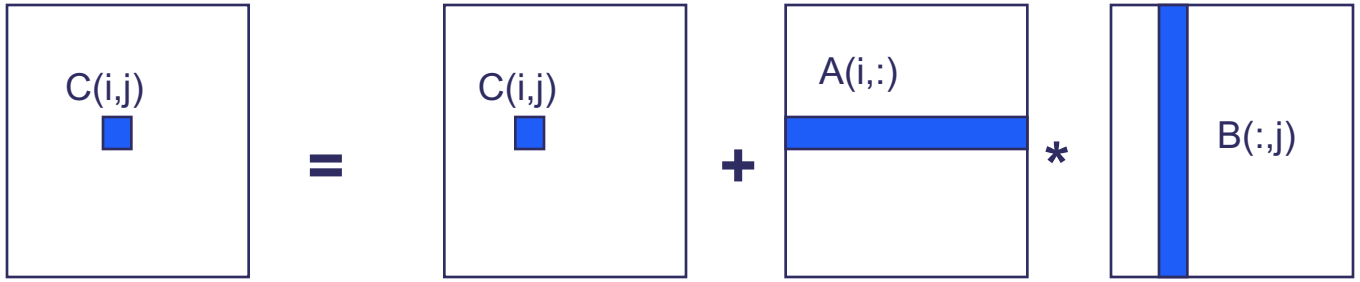
```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}
   for j = 1 to n
      {read C(i,j) into fast memory}
      {read column j of B into fast memory}
      for k = 1 to n
         C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write C(i,j) back to slow memory}
```



Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}              ... n^2 reads altogether
   for j = 1 to n
      {read C(i,j) into fast memory}              ... n^2 reads altogether
      {read column j of B into fast memory}       ... n^3 reads altogether
      for k = 1 to n
         C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write C(i,j) back to slow memory}          ... n^2 writes altogether
```

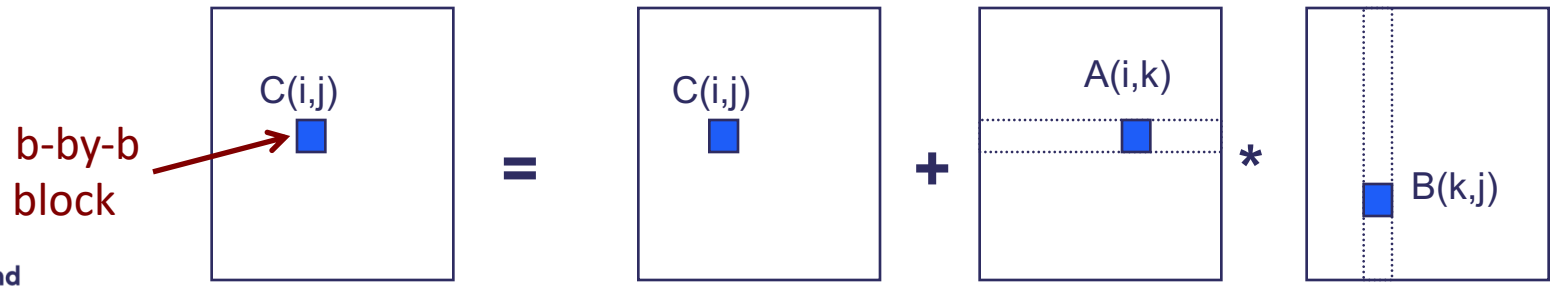


$n^3 + 3n^2$ reads/writes altogether – dominates $2n^3$ arithmetic

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where b is called the block size; assume 3 b-by-b blocks fit in fast memory

```
for i = 1 to n/b
  for j = 1 to n/b
      {read block C(i,j) into fast memory}
      for k = 1 to n/b
          {read block A(i,k) into fast memory}
          {read block B(k,j) into fast memory}
          C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
      {write block C(i,j) back to slow memory}
```

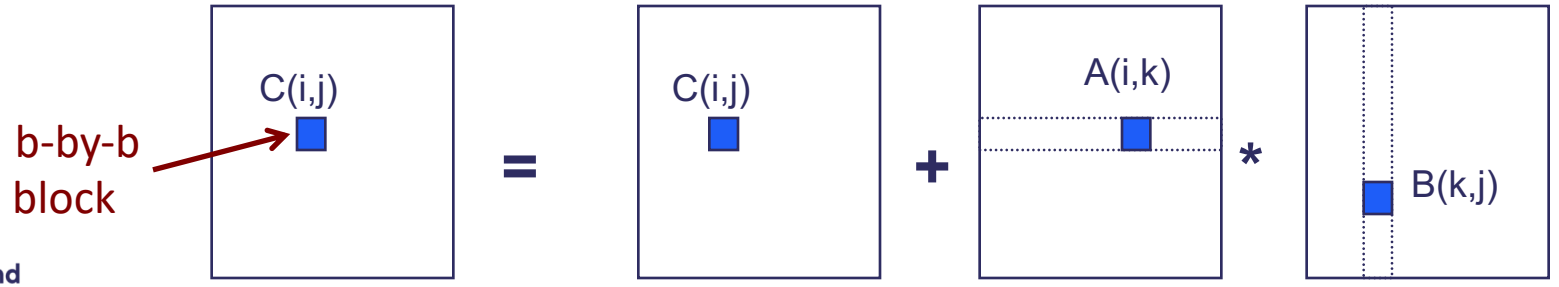


Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where b is called the block size; assume 3 b-by-b blocks fit in fast memory

```
for i = 1 to n/b
  for j = 1 to n/b
    {read block C(i,j) into fast memory}          ... b² × (n/b)² = n² reads
    for k = 1 to n/b
      {read block A(i,k) into fast memory}        ... b² × (n/b)³ = n³/b reads
      {read block B(k,j) into fast memory}        ... b² × (n/b)³ = n³/b reads
      C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory}      ... b² × (n/b)² = n² writes
```



$2n^3/b + 2n^2$ reads/writes << $2n^3$ arithmetic - Faster!

Does blocked matmul attain lower bound?

- Recall: if 3 b-by-b blocks fit in fast memory of size M, then #reads/writes = $2n^3/b + 2n^2$
- Make b as large as possible: $3b^2 \leq M$, so
- #reads/writes $\geq 3^{1/2}n^3/M^{1/2} + 2n^2$
- Attains lower bound = Ω (#flops / $M^{1/2}$)

- But what if we don't know M?
- Or if there are multiple levels of fast memory?
- How do we write the algorithm?

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Parallel MatMul with 2D Processor Layout

- P processors in $P^{1/2}$ x $P^{1/2}$ grid
  - Processors communicate along rows, columns
- Each processor owns $n/P^{1/2}$ x $n/P^{1/2}$ submatrices of A,B,C
- Example: P=16, processors numbered from P_{00} to P_{33}
  - Processor P_{ij} owns submatrices A_{ij}, B_{ij} and C_{ij}

C = A * B

C:

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

A:

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

B:

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

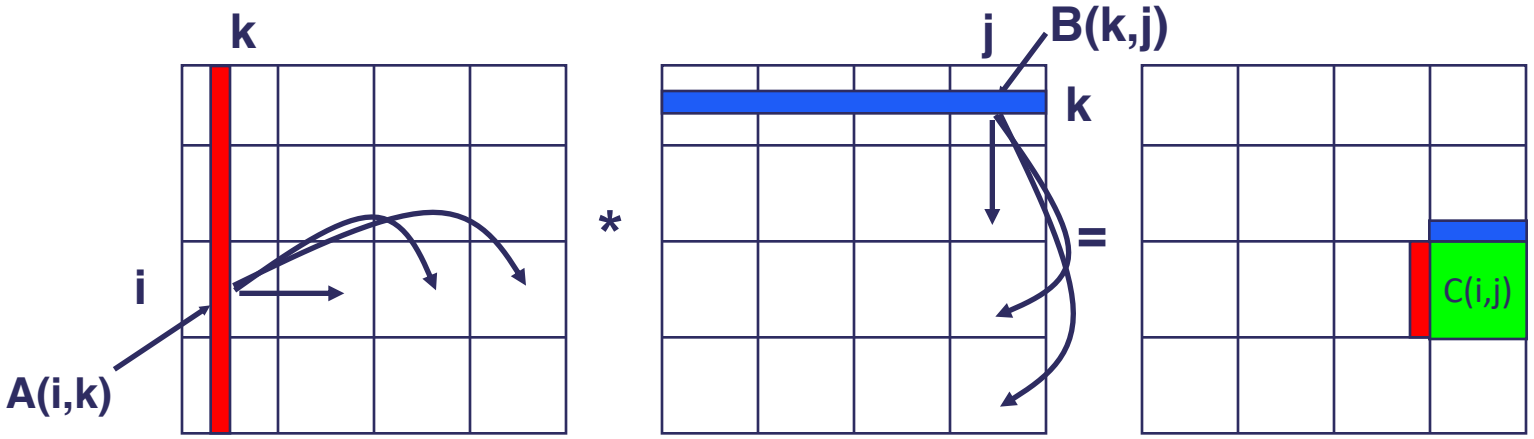
SUMMA Algorithm

- SUMMA = Scalable Universal Matrix Multiply
  - Comes within factor of log P of lower bounds:
    - Assume fast memory size M = $O(n^2/P)$ per processor – 1 copy of data
    - #words_moved = $\Omega(\ \#flops / M^{1/2}\ ) = \Omega(\ (n^3/P) / (n^2/P)^{1/2}\ ) = \Omega(\ n^2 / P^{1/2}\ )$
    - #messages = $\Omega(\ \#flops / M^{3/2}\ ) = \Omega(\ (n^3/P) / (n^2/P)^{3/2}\ ) = \Omega(\ P^{1/2}\ )$
  - Can accommodate any processor grid, matrix dimensions & layout
  - Used in practice in PBLAS = Parallel BLAS

- Comparison to Cannon's Algorithm
  - Cannon attains lower bound
  - But Cannon harder to generalize to other grids, dimensions, layouts, and Cannon may use more memory

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/59C12_tutorial

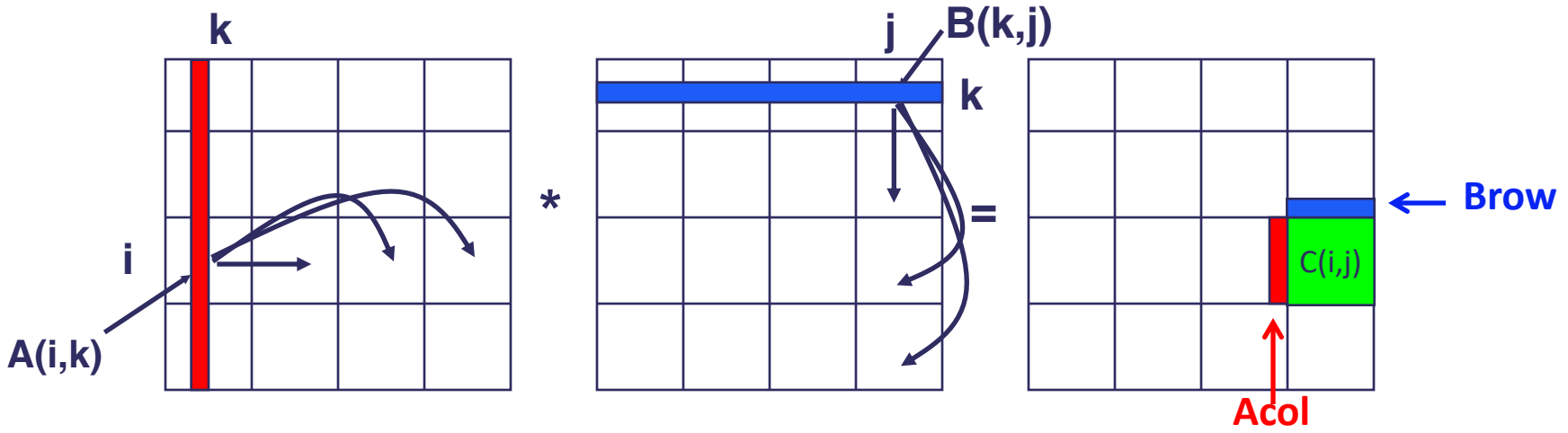
SUMMA – n x n matmul on $P^{1/2}$ x $P^{1/2}$ grid



- C(i, j) is $n/P^{1/2}$ x $n/P^{1/2}$ submatrix of C on processor P_{ij}
- A(i,k) is $n/P^{1/2}$ x b submatrix of A
- B(k,j) is b x $n/P^{1/2}$ submatrix of B
- C(i,j) = C(i,j) + Σ_k A(i,k)*B(k,j)
  - summation over submatrices
- Need not be square processor grid

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

SUMMA– n x n matmul on $P^{1/2}$ x $P^{1/2}$ grid



For k=0 to n/b-1

for all i = 1 to $P^{1/2}$

owner of A(i,k) broadcasts it to whole processor row (using binary tree)

for all j = 1 to $P^{1/2}$

owner of B(k,j) broadcasts it to whole processor column (using bin. tree)

Receive A(i,k) into Acol

Receive B(k,j) into Brow

C_myproc = C_myproc + Acol * Brow

SUMMA Communication Costs

For k=0 to n/b-1
  for all i = 1 to $P^{1/2}$
    owner of A(i,k) broadcasts it to whole processor row (using binary tree)
    … #words = log $P^{1/2}$ *b*n/$P^{1/2}$, #messages = log $P^{1/2}$
  for all j = 1 to $P^{1/2}$
    owner of B(k,j) broadcasts it to whole processor column (using bin. tree)
    … same #words and #messages
  Receive A(i,k) into Acol
  Receive B(k,j) into Brow
  C_myproc = C_myproc + Acol * Brow

- Total #words = log P * n^2 /$P^{1/2}$
  - Within factor of log P of lower bound
  - (more complicated implementation removes log P factor)
- Total #messages = log P * n/b
  - Choose b close to maximum, n/$P^{1/2}$, to approach lower bound $P^{1/2}$

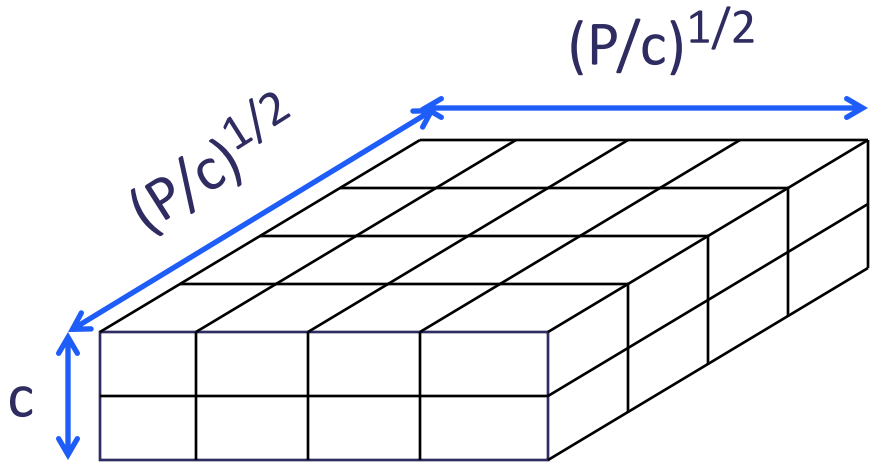
Can we do better?

- Lower bound assumed 1 copy of data: $M = O(n^2/P)$ per proc.
- What if matrix small enough to fit c>1 copies, so $M = cn^2/P$?
  - #words_moved = $\Omega(\ \#flops / M^{1/2}\ ) = \Omega(\ n^2 / (\ c^{1/2}\ P^{1/2}\ ))$
  - #messages = $\Omega(\ \#flops / M^{3/2}\ ) = \Omega(\ P^{1/2}\ /c^{3/2})$
- Can we attain new lower bound?
  - Special case: "3D Matmul": $c = P^{1/6}$
    - Dekel, Nassimi, Sahni [81], Bernsten [89], Agarwal, Chandra, Snir [90], Johnson [93], Agarwal, Balle, Gustavson, Joshi, Palkar [95]
    - Processors arranged in $P^{1/3} \times P^{1/3} \times P^{1/3}$ grid
    - Processor (i,j,k) performs C(i,j) = C(i,j) + A(i,k)*B(k,j), where each submatrix is $n/P^{1/3} \times n/P^{1/3}$
  - Not always that much memory available…

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

2.5D Matrix Multiplication

- Assume can fit cn^2/P data per processor, $c>1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid

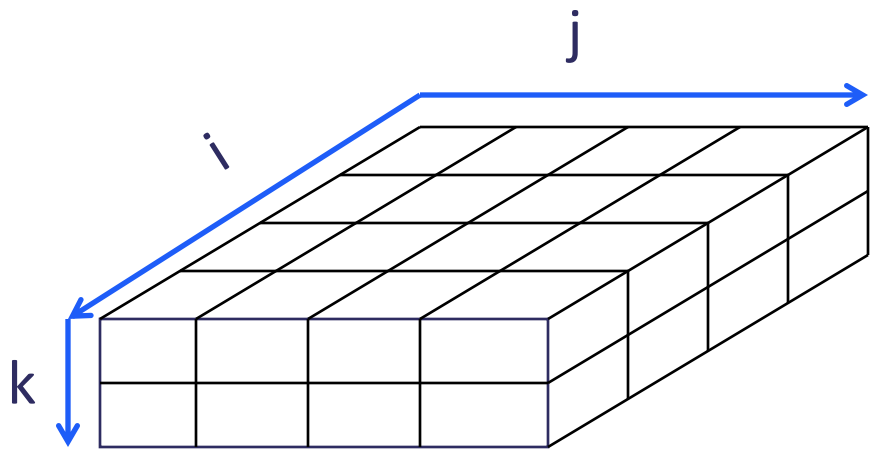


Example: P = 32, c = 2

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

2.5D Matrix Multiplication

- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid

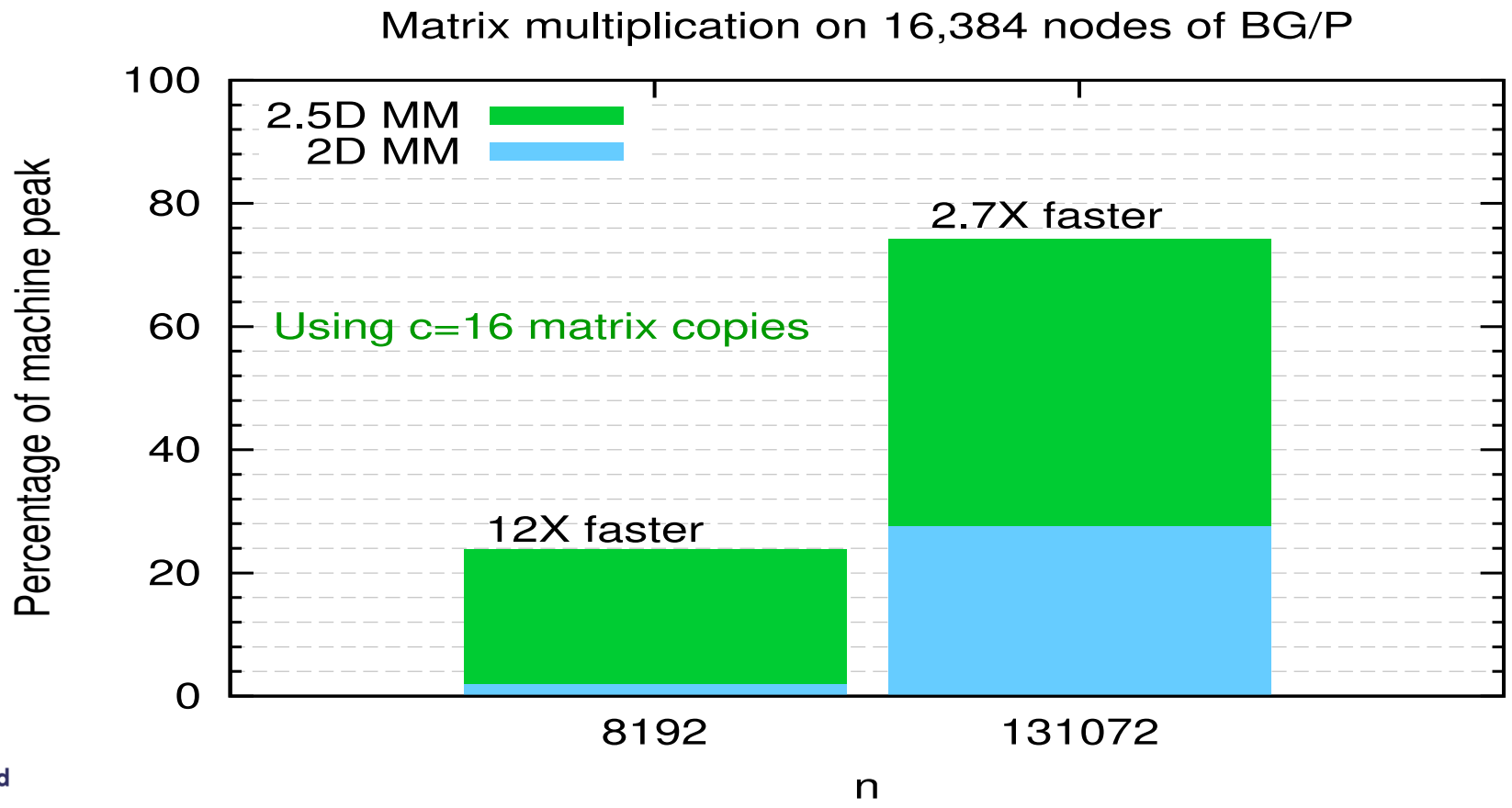


Initially P(i,j,0) owns A(i,j) and B(i,j)
each of size $n(c/P)^{1/2}$ x $n(c/P)^{1/2}$

(1) P(i,j,0) broadcasts A(i,j) and B(i,j) to P(i,j,k)
(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of Σ_m A(i,m)*B(m,j)
(3) Sum-reduce partial sums Σ_m A(i,m)*B(m,j) along k-axis so P(i,j,0) owns C(i,j)

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

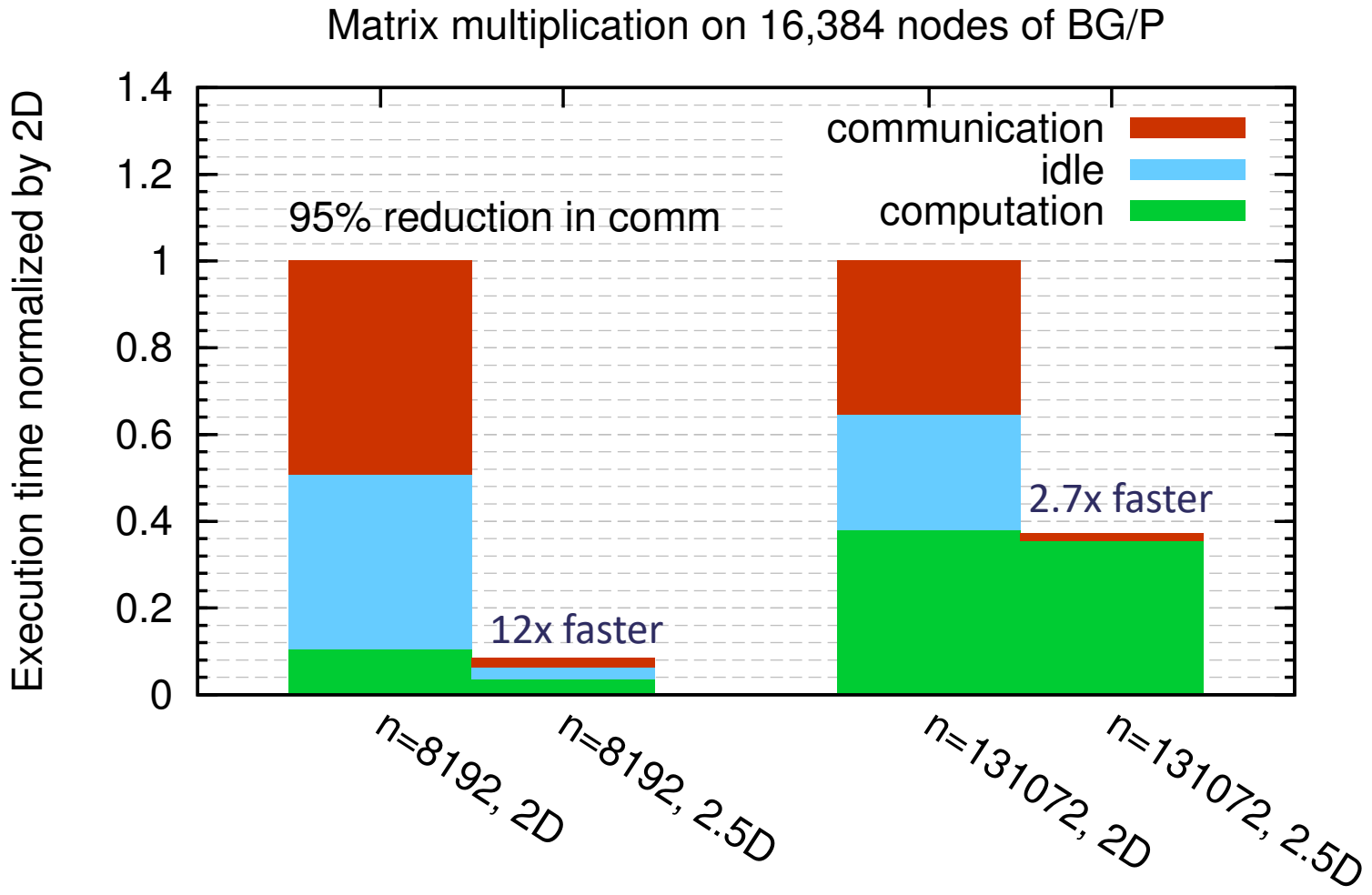
2.5D Matmul on BG/P, 16K nodes / 64K cores



Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

2.5D Matmul on BG/P, 16K nodes / 64K cores

c = 16 copies



Science and Technology Facilities Council
Hartree Centre

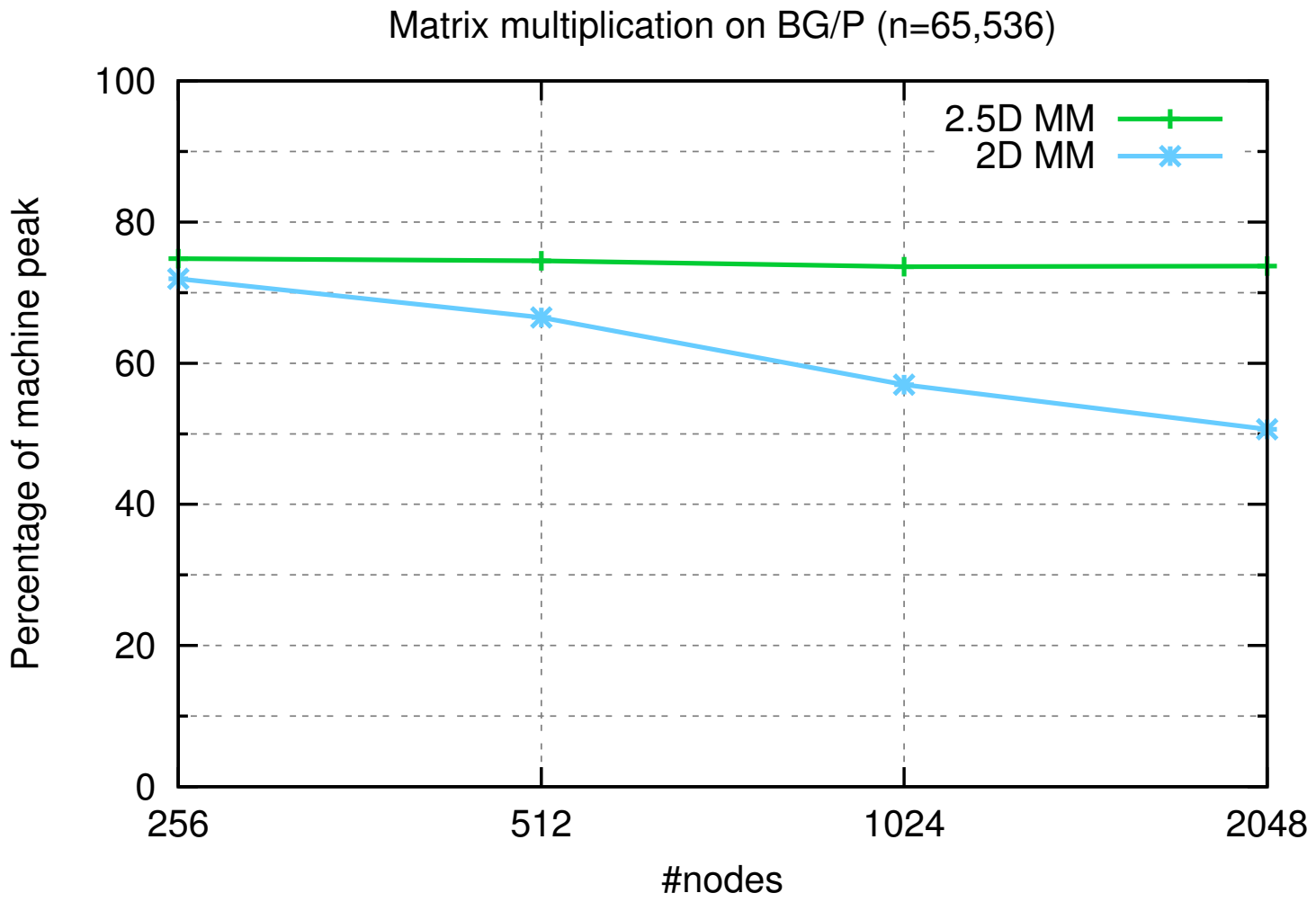
Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Perfect Strong Scaling – in Time and Energy (1/2)

- Every time you add a processor, you should use its memory M too
- Start with minimal number of procs: $PM = 3n^2$
- Increase P by a factor of c ➔ total memory increases by a factor of c
- Notation for timing model:
  - γ_T , β_T , α_T = secs per flop, per word_moved, per message of size m
- $T(cP) = n^3/(cP) [\gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})]$
  $= T(P)/c$
- Notation for energy model:
  - γ_E , β_E , α_E = joules for same operations
  - δ_E = joules per word of memory used per sec
  - ε_E = joules per sec for leakage, etc.
- $E(cP) = cP \{ n^3/(cP) [\gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})] + \delta_E MT(cP) + \varepsilon_E T(cP) \}$
  $= E(P)$

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Perfect Strong Scaling in Time of 2.5D Matmul on BG/P, n=64K



Science and Technology Facilities Council
Hartree Centre

Perfect Strong Scaling – in Time and Energy (2/2)

We can use these models to answer many questions, including:

- What is the minimum energy required for a computation?
- Given a maximum allowed runtime **T** , what is the minimum energy **E** needed to achieve it?
- Given a maximum energy budget **E** , what is the minimum runtime **T** that we can attain?
- The ratio **P** = **E/T** gives us the average power required to run the algorithm. Can we minimize the average power consumed?
- Given an algorithm, problem size, number of processors and target energy efficiency (GFLOPS/W), can we determine a set of architectural parameters to describe a conforming computer architecture?

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

Recall optimal sequential Matmul

- Naïve code
  for i=1:n, for j=1:n, for k=1:n, C(i,j)+=A(i,k)*B(k,j)

- "Blocked" code
  for i1 = 1:b:n, for j1 = 1:b:n, for k1 = 1:b:n
  for i2 = 0:b-1, for j2 = 0:b-1, for k2 = 0:b-1
  i=i1+i2, j = j1+j2, k = k1+k2
  C(i,j)+=A(i,k)*B(k,j)

  } b x b matmul

- Thm: Picking b = $M^{\mathbf{1/2}}$ attains lower bound:
  #words_moved = $\Omega(n^3/M^{\mathbf{1/2}})$
- Where does **1/2** come from?

Science and Technology Facilities Council
Hartree Centre

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

New Thm applied to Matmul

for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)

Record array indices in matrix Δ

$$\Delta = \begin{array}{c} \begin{matrix} i & j & k \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} \\ A \\ B \\ C \end{matrix}$$

Solve LP for $x = [xi,xj,xk]^T$: max $\mathbf{1}^T x$ s.t. $\Delta x \le \mathbf{1}$

Result: $x = [1/2, 1/2, 1/2]^T$, $\mathbf{1}^T x = 3/2 = s$

Thm: #words_moved = $\Omega(n^3/M^{S-1}) = \Omega(n^3/M^{1/2})$

Attained by block sizes $M^{xi},M^{xj},M^{xk} = M^{1/2},M^{1/2},M^{1/2}$

Some material from Prof. Jim Demmel www.cs.berkeley.edu/~demmel/SC12_tutorial

For more details and communication avoiding algorithms for other type of problems please see:

- Bebop.cs.berkeley.edu
- https://www.youtube.com/watch?v=2vt502jlNbY
- www.cs.berkeley.edu/~demmel

Science and Technology Facilities Council

Hartree Centre

Comfort Break

Science and Technology Facilities Council

Hartree Centre

Science and Technology Facilities Council

Hartree Centre

Thank you

vassil.alexandrov@stfc.ac.uk
dominic.richards@stfc.ac.uk

 hartree.stfc.ac.uk

 @HartreeCentre

 STFC Hartree Centre

 hartree@stfc.ac.uk