Data Aware Algorithms – Part 1

Loris Marchal

October 17, 2023

Data Aware Algorithms

Topics covered:

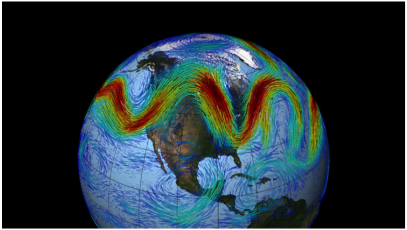
- Pebble game models
- I/Os lower bounds
- Communication-avoiding algorithms
- Cache oblivious algorithms
- Memory-aware scheduling

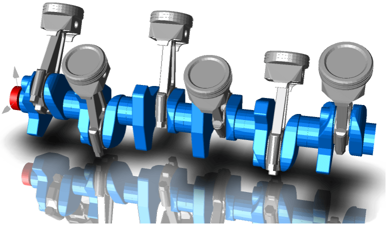
Contact: loris.marchal@ens-lyon.fr

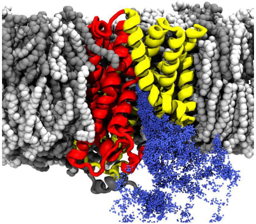
More material:

http://perso.ens-lyon.fr/loris.marchal/data-aware-algorithms-warsaw.html

High Performance Computing



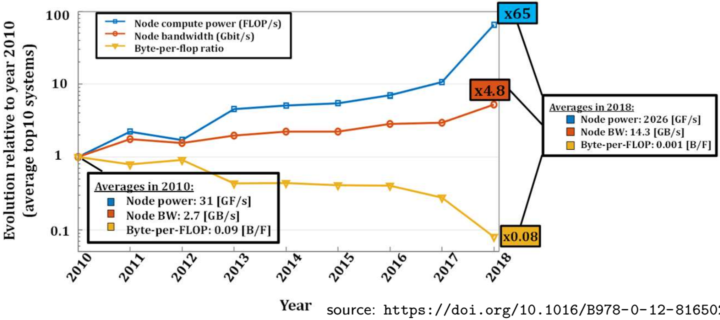




- Numerical simulations drive new discoveries
- Larger systems with better accuracy: more data and computation

Data access problem

Evolution of computing speed vs. data access speed (bandwidth)

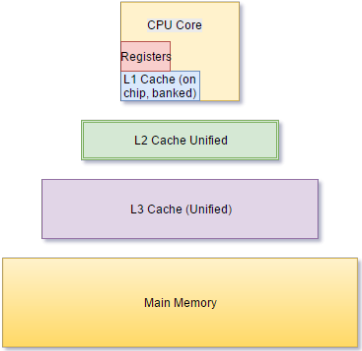


source: `https://doi.org/10.1016/B978-0-12-816502-7.00020-8`

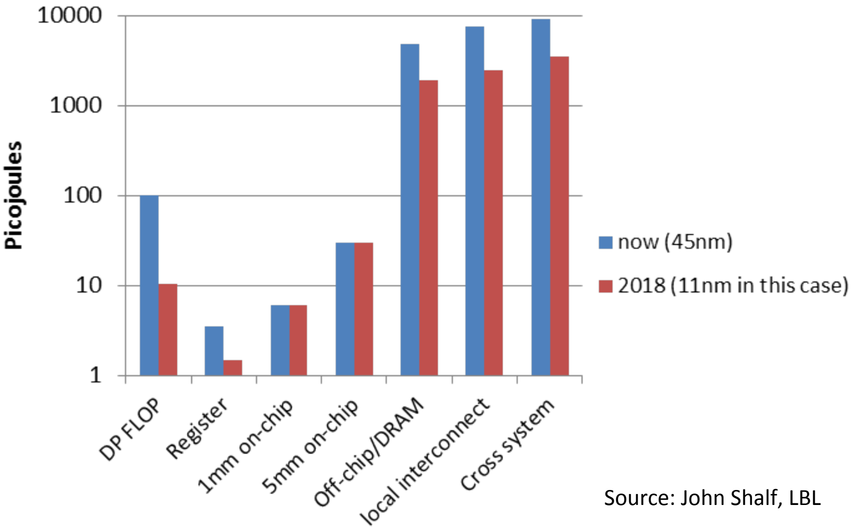
Byte-per-flop ratio keeps decreasing $\Rightarrow$ Data access critical for performance

Beyond the memory wall

- Time to move the data > Time to compute on the data
- Similar problem in microprocessor design: “memory wall”
- Traditional workaround:
  add a faster but smaller “cache” memory
- Now a hierarchy of caches !



Energy required for communications



Source: John Shalf, LBL

Computing with bounded cache/memory

- Limited amount of fast cache
- Performance sensitive to data locality
- Optimize data reuse
- Avoid data movements (I/Os) between memory and cache(s) (time-consuming and energy-consuming)

In this talk: some algorithmic approaches to this problem

Computing with bounded cache/memory

- Limited amount of fast cache
- Performance sensitive to data locality
- Optimize data reuse
- Avoid data movements (I/Os) between memory and cache(s) (time-consuming and energy-consuming)

In this talk: some algorithmic approaches to this problem

Data Aware Algorithms – Part 1

Data Aware Algorithms – Part 1

Pebble game models

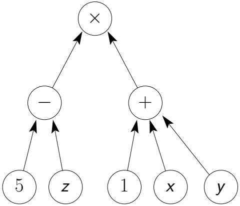
Algorithm Design and Data Movement: the Matrix Product Case

Analysis and Lower Bounds for Parallel Algorithms

Conclusion

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

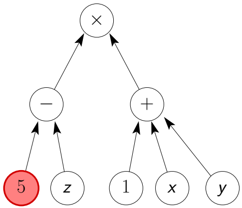
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$$(5 - z) \times (1 + x + y)$$

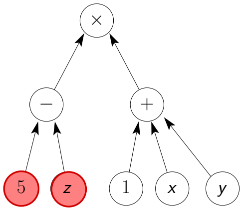
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

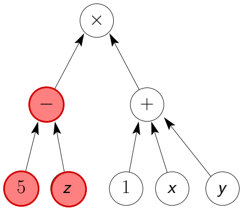
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

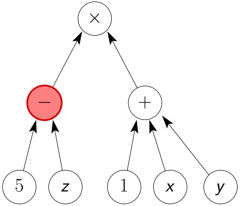
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

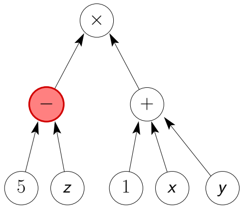
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

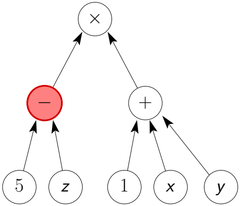
Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble game for register allocation

- From the 70s: limit usage of scarce registers
- Model expressions as Directed Acyclic Graphs



$(5 - z) \times (1 + x + y)$

Rules of the game:

- A pebble may be placed on a source node at any time (LOAD)
- If all predecessors of v are pebbled, a pebble may be placed on v. (COMPUTE)
- A pebble may be removed from a vertex at any time. (EVICT)
- Goal: computation all vertices, use minimal number of pebbles

Results: Optimal algorithms for trees — NP-hard on general DAGs

Pebble Game – Complexity, variants, space-time tradeoffs

Progressive pebble game:

- Forbid pebbling twice the same vertex, NP-Hard

More general problem with re-computation:

- PSpace-complete

Variant with pebble shifting:

- Rule 3 $\rightarrow$ If all predecessors of an unpebbled vertex v are pebbled, a pebble may be shifted from a predecessor to v.

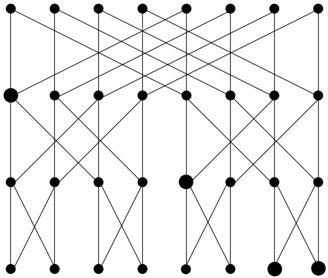
Space-Time Tradoffs – Example

Every pebbling strategy for any program computing the multiplication of two $N \times N$ matrices uses a space S and time T respecting the following inequality:

$$(S+1)T \geq N^3/4$$

Space-Time tradeoffs – FFT example

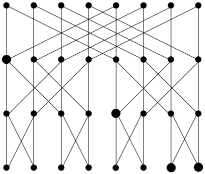
- Fast-Fourrier Transform
- Recursive graph based on the “exchange graph” with 2 inputs and 2 outputs



FFT graph with 8 input/output vertices (depth $k = 3$)
$n = 2^k$ vertices at each level

- Strategy minimizing the computation cost? the memory?

Space-Time tradeoffs – FFT example



Strategy 1:

- Pebble level by level
- Requires $2n = 2^{k+1}$ pebbles (or $n + 2$ if done carefully)
- No recomputations (minimum number of steps)

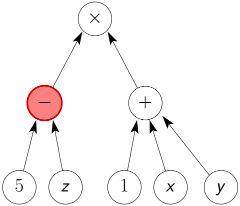
Strategy 2:

- Pebble one tree up to one output, then start over
  (variant: pebble two outputs before re-starting)
- Uses $k + 1$ pebbles
  (minimum value since it contains binary tree of depth k)
- Large number of recomputations

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)



Goal: minimize number of WRITE

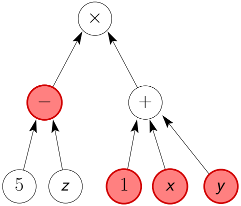
Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)



Goal: minimize number of WRITE

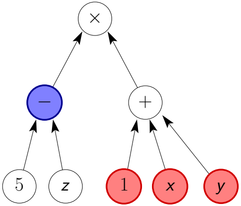
Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)



Goal: minimize number of WRITE

Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

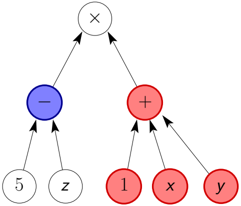
When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)

Goal: minimize number of WRITE

Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

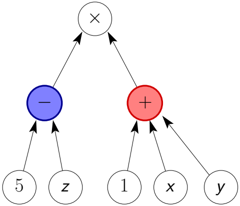


- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)



Goal: minimize number of WRITE

Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

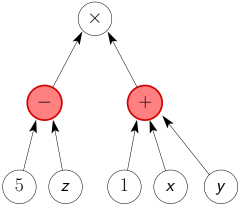
- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)

Goal: minimize number of WRITE



Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

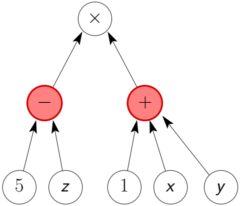
- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

When memory too limited: minimize I/Os

Red/Blue pebble game [Hong & Kung, 1981]
New rules:

- Limited number of red pebbles (=memory slots)
- Replace red pebble by blue pebble (WRITE)
- Replace blue pebble by red pebble (READ)

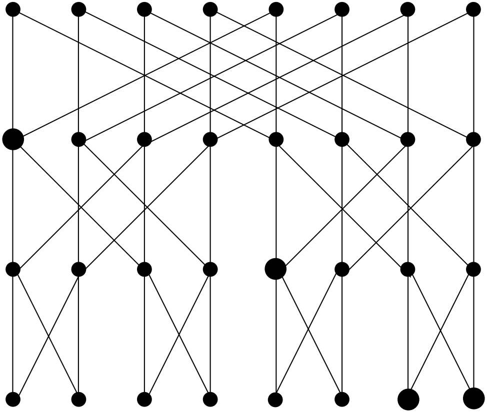
Goal: minimize number of WRITE



Model applies to any two-memory system:
(fast, bounded) memory vs. (slow, large) disk

- Successful to design **lower bounds on I/Os** and optimal algorithms
- Basis for other studies: communication-avoiding algorithms (recomputations may be allowed or forbidden)

Example: FFT graph



k levels, $n = 2^k$ vertices at each level

Minimum number S of red pebbles ?

How many I/Os for this minimum number S ?

Data Aware Algorithms – Part 1

Pebble game models

Algorithm Design and Data Movement: the Matrix Product Case

Analysis and Lower Bounds for Parallel Algorithms

Conclusion

Example: matrix-matrix product

- Consider two square matrices A and B (size $n \times n$)
- Compute generalized matrix product: $C \leftarrow C + AB$

```
Simple-Matrix-Multiply(n, C, A, B)
for i = 0 → n − 1 do
    for j = 0 → n − 1 do
        for k = 0 → n − 1 do
            C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}
```

Assume simple two-level memory model:

- Slow but infinite disk storage (where A and B are originally stored)
- Fast and limited memory (size M)

Objective: limit data movement between disk/memory

NB: also applies to other two-level systems (memory/cache, etc.)

Simple algorithm analysis

Simple-Matrix-Multiply(n, C, A, B)
for $i = 0 \to n-1$ **do**
  for $j = 0 \to n-1$ **do**
    for $k = 0 \to n-1$ **do**
      $C_{i,j} = C_{i,j} + A_{i,k}B_{k,j}$

- Assume the memory cannot store half of a matrix: $M < n^2/2$
- Question: How many data movement in this algorithm ?

Simple algorithm analysis

Simple-Matrix-Multiply(n, C, A, B)

for $i = 0 \to n-1$ **do**
  for $j = 0 \to n-1$ **do**
    for $k = 0 \to n-1$ **do**
      $C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}$

- Assume the memory cannot store half of a matrix: $M < n^2/2$
- Question: How many data movement in this algorithm ?

Answer:

- all elements of B accessed during one iteration of the outer loop
- At most half of B stays in memory
- At least $n^2/2$ elements must be read per outer loop
- At least $n^3/2$ read for entire algorithms
- Same order of magnitude of computations: $O(n^3)$
- Very bad data reuse ☹ Question: How to do better ?

Blocked matrix-matrix product

- Divide each matrix into blocks of size $b \times b$:
  $A^b_{i,k}$ is the block of A at position (i, k)
- Perform "coarse-grain" matrix product on blocks
- Perform each block product with previous algorithms

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply($n, C^b_{i,j}, A^b_{i,k}, B^b_{k,j}$)

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)

$b \leftarrow \sqrt{M/3}$

for $i = 0, \rightarrow n/b - 1$ **do**

  for $j = 0, \rightarrow n/b - 1$ **do**

    for $k = 0, \rightarrow n/b - 1$ **do**

      Simple-Matrix-Multiply$(n, C^b_{i,j}, A^b_{i,k}, B^b_{k,j})$

Question: Number of data movements ?

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply($n, C^b_{i,j}, A^b_{i,k}, B^b_{k,j}$)

Question: Number of data movements ?

- Iteration of inner loop: 3 blocks of size $b \times b = \sqrt{M/3}^3 = M/3$ $\rightarrow$ fits in memory
- At most $M + M/3 = O(M)$ data movements for each inner loop (reading/writing)
- Number of inner iterations: $(n/b)^3 = O(n^3/M^{3/2})$
- Total number of data movements: $O(n^3/\sqrt{M})$

Blocked matrix-matrix product – Analysis

Blocked-Matrix-Multiply(n,A,B,C)
$b \leftarrow \sqrt{M/3}$
for $i = 0, \rightarrow n/b - 1$ **do**
  for $j = 0, \rightarrow n/b - 1$ **do**
    for $k = 0, \rightarrow n/b - 1$ **do**
      Simple-Matrix-Multiply($n, C^b_{i,j}, A^b_{i,k}, B^b_{k,j}$)

Question: Number of data movements ?

- Iteration of inner loop: 3 blocks of size $b \times b = \sqrt{M/3}^3 = M/3$ $\rightarrow$ fits in memory
- At most $M + M/3 = O(M)$ data movements for each inner loop (reading/writing)
- Number of inner iterations: $(n/b)^3 = O(n^3/M^{3/2})$
- Total number of data movements: $O(n^3/\sqrt{M})$

Question: Can we do (significantly) better ?

I/O lower bound for matrix multiplication

Theorem (Hong& Kung 1981, Toledo 1999).

Any conventional matrix multiplication algorithm will perform at least $\Omega(n^3/\sqrt{M})$ I/O operations.

conventional: perform all n^3 elementary products
(aka: not Strassen or Coppersmith-Winograd)

I/O lower bound for matrix multiplication – proof 1/2

- Decompose the computation into phases of M I/O operations (except the last phase, which may contain $< M$ operations
- $C_{i,j}$ is live in a phase if some $A_{i,k} \times B_{k,j}$ is computed
- During a phase:
  - At most $2M$ elements of A are available for computations: A_p (M from the memory, M from reads)
  - Same for B ($|B_p| \leq 2M$)
  - At most $2M$ "live" $C_{i,j}$ (M in memory at the end, M written during the phase)

Goal: bound the number of elementary matrix products done in one phase

I/O lower bound for matrix multiplication – proof 2/2

Two cases for elements of A_p:

- Dense rows of A_p
  - S_p^1: set of rows of A with at least $\sqrt{M}$ elements in A_p, $|S_p^1| \leq 2\sqrt{M}$
    Each element of B_p multiplied by at most one element from each row of S_p^1
  - At most $2\sqrt{M} \times 2M = 4M^{3/2}$ multiplications with elements from S_p^1
- Sparse rows of A_p
  - Each "live" $C_{i,j}$ = one row of A $\times$ one column of B
    Number of elementary product for each $C_{i,j} \leq$ size of the corresponding row
  - For sparse rows ($\notin S_1^p$), at most $2M \times \sqrt{M}$ products

Overall, at most $6M^{3/2}$ elementary products per phase.

Total number of full phases $\geq \lfloor \frac{n^3}{6M^{3/2}} \rfloor - 1 \geq \frac{n^3}{6M^{3/2}} - 1$

Total number of I/Os $\geq \frac{n^3}{6\sqrt{M}} - M$

I/O lower bound for matrix multiplication – proof 2/2

Two cases for elements of A_p:

- Dense rows of A_p
  - S_p^1: set of rows of A with at least $\sqrt{M}$ elements in A_p, $|S_p^1| \leq 2\sqrt{M}$
    Each element of B_p multiplied by at most one element from each row of S_p^1
  - At most $2\sqrt{M} \times 2M = 4M^{3/2}$ multiplications with elements from S_p^1
- Sparse rows of A_p
  - Each "live" $C_{i,j}$ = one row of A × one column of B
    Number of elementary product for each $C_{i,j} \leq$ size of the corresponding row
  - For sparse rows ($\notin S_1^p$), at most $2M \times \sqrt{M}$ products

Overall, at most $6M^{3/2}$ elementary products per phase.

Total number of full phases $\geq \lfloor \frac{n^3}{6M^{3/2}} \rfloor - 1 \geq \frac{n^3}{6M^{3/2}} - 1$

Total number of I/Os $\geq \frac{n^3}{6\sqrt{M}} - M$

I/O lower bound for matrix multiplication – proof 2/2

Two cases for elements of A_p:

- Dense rows of A_p
  - S_p^1: set of rows of A with at least $\sqrt{M}$ elements in A_p, $|S_p^1| \leq 2\sqrt{M}$
    Each element of B_p multiplied by at most one element from each row of S_p^1
  - At most $2\sqrt{M} \times 2M = 4M^{3/2}$ multiplications with elements from S_p^1
- Sparse rows of A_p
  - Each "live" $C_{i,j}$ = one row of A $\times$ one column of B
    Number of elementary product for each $C_{i,j} \leq$ size of the corresponding row
  - For sparse rows ($\notin S_1^p$), at most $2M \times \sqrt{M}$ products

Overall, at most $6M^{3/2}$ elementary products per phase.

Total number of full phases $\geq \lfloor \frac{n^3}{6M^{3/2}} \rfloor - 1 \geq \frac{n^3}{6M^{3/2}} - 1$

Total number of I/Os $\geq \frac{n^3}{6\sqrt{M}} - M$

Tight Lower Bound for Matrix Product

```
b ← √(M/3)
for i = 0, → n/b − 1 do
    for j = 0, → n/b − 1 do
        for k = 0, → n/b − 1 do
            Simple-Matrix-Multiply(n, C^b_{i,j}, A^b_{i,k}, B^b_{k,j})
```

- I/Os of blocked algorithm: $2\sqrt{3}N^3/\sqrt{M} + N^2$
- Lower bound on I/Os $\sim N^3/6\sqrt{M}$
- Many improvements needed to close the gap
- Presented here for $C \leftarrow C + AB$, square matrices

New operation: Fused Multiply Add

- Perform $c \leftarrow c + a \times b$ in a single step
- No temporary storage needed (3 inputs, 1 output)

Step 1: Use Only FMAs (Fused Multiply Add)

Theorem.

Any algorithm for the matrix product can be transformed into using only FMA without increasing the required memory or the number of I/Os.

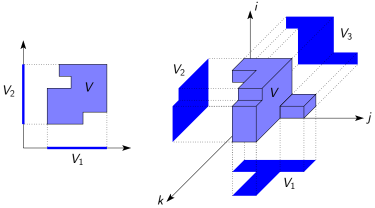
Transformation:

- If some $c_{i,j,k}$ is computed while $c_{i,j}$ is not in memory, insert a read before the multiplication
- Replace the multiplication by a FMA
- Remove the read that must occur before the addition $c_{i,j} \leftarrow c_{i,j} + c_{i,j,k}$, remove the addition
- Transform occurrences of $c_{i,j,k}$ into $c_{i,j}$
- If $c_{i,j,k}$ and $c_{i,j}$ were both in memory in some time-interval, remove operations with $c_{i,j,k}$ in this interval

Step 2: Concentrate on Read Operations

Theorem (Irony, Toledo, Tiskin, 2008).

Using N_A elements of A, N_B elements of B and N_C elements of C, we can perform at most $\sqrt{N_A N_B N_C}$ distinct FMAs.



Theorem (Discrete Loomis-Whitney Inequality).

Let V be a finite subset of $\mathbb{Z}^3$ and V_1, V_2, V_3 denotes the orthogonal projections of V on each coordinate planes, we have

$$|V|^2 \leq |V_1| \cdot |V_2| \cdot |V_3|,$$

Step 3: Use Phases of R Reads ($\neq M$)

Theorem.

During a phase with R reads with memory M, the number of FMAs is bounded by

$$F_{M+R} \leq \left(\frac{1}{3}(M+R)\right)^{3/2}$$

Number F_{M+R} of FMAs constrained by:

$$\begin{cases} F_{M+R} \leq \sqrt{N_A N_B N_C} \\ 0 \leq N_A, N_B, N_C \\ N_A + N_B + N_C \leq M + R \end{cases}$$

Using Lagrange multipliers, maximal value obtained when $N_A = N_B = N_C$

Step 4: Choose R and add write operations

in one phase, nb of computations: $F_{M+R} \leq \left(\frac{1}{3}(M+R)\right)^{3/2}$

Total volume of reads:

$$V_{\text{read}} \geq \left\lfloor \frac{N^3}{F_{M+R}} \right\rfloor \times R \geq \left(\frac{N^3}{F_{M+R}} - 1\right) \times R$$

Valid for all values of R, maximized when $R = 2M$:

$$V_{\text{read}} \geq 2N^3/\sqrt{M} - 2M$$

Each element of C written at least once: $V_{\text{write}} \geq N^2$

Theorem.

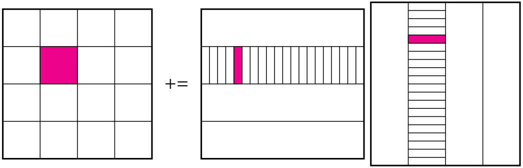
The total volume of I/Os is bounded by:

$$V_{I/O} \geq \frac{2N^3}{\sqrt{M}} + N^2 - 2M$$

Exercise: asymptotically optimal algorithm

Consider the following algorithm sketch:

- Partition C into blocks of size $(\sqrt{M}-1) \times (\sqrt{M}-1)$
- Partition A into block-columns of size $(\sqrt{M}-1) \times 1$
- Partition B into block-rows of size $1 \times (\sqrt{M}-1)$
- For each block C_b of C:
  - Load the corresponding blocks of A and B on after the other
  - For each pair of blocks A_b, B_b, compute $C_b \leftarrow C_b + A_b B_b$
  - When all products for C_b are performed, write back C_b



1. Write a proper algorithm following these directions
2. Compute the number of read and write operations

Generalization to other Linear Algebra Algorithms

Theorem (Ballard et al., 2011).

For any matrix computation expressed as "general computations", the number of I/Os is at least $G/(8\sqrt{M}) - M$, where G is the total number of elementary operations g.

General computation

For all $(i,j) \in S_c$,

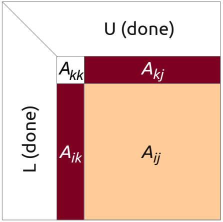
$$C_{i,j} \leftarrow f_{i,j}\Big(g_{i,j,k}(A_{i,k}B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments}\Big)$$

- f_i, j and $g_{i,j,k}$ must be "non-trivial"
- For matrix multiplication:
  - $f_{i,j}$: summation, $g_{i,j,k}$: product
  - $S_{i,j} = [1,n]$, $S_C = [1,n] \times [1,n]$

Application to LU Factorization (1/2)

LU factorization (Gaussian elimination):

- Convert a matrix A into product $L \times U$
- L is lower triangular with diagonal 1
- U is upper triangular
- $(L - D + U)$ stored in place with A



LU Algorithm

For $k = 1 \dots n-1$:

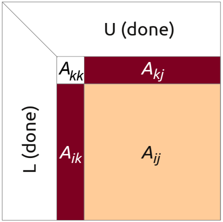
- For $i = k+1 \dots n$,
  $A_{i,k} \leftarrow a_{i,k}/a_{k,k}$ (column/panel preparation)
- For $i = k+1 \dots n$,
  For $j = k+1 \dots n$,
  $A_{i,j} \leftarrow A_{i,j} - A_{i,k}A_{k,j}$ (update)

Application to LU Factorization (2/2)

Can be expressed as follows:

$$U_{i,j} = A_{i,j} - \sum_{k<i} L_{i,k} \cdot U_{k,j} \qquad \text{for } i \leq j$$

$$L_{i,j} = (A_{i,j} - \sum_{k<j} L_{i,k} \cdot U_{k,j}) / U_{j,j} \qquad \text{for } i > j$$



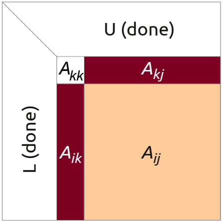
Fits the generalized matrix computations:

$$C(i,j) = f_{i,j}\left(g_{i,j,k}(A(i,k), B(k,j)) \text{ for } k \in S_{i,j}, K\right)$$

with:

Application to LU Factorization (2/2)

Can be expressed as follows:



$$U_{i,j} = A_{i,j} - \sum_{k<i} L_{i,k} \cdot U_{k,j} \qquad \text{for } i \leq j$$

$$L_{i,j} = (A_{i,j} - \sum_{k<j} L_{i,k} \cdot U_{k,j}) / U_{j,j} \qquad \text{for } i > j$$

Fits the generalized matrix computations:

$$C(i,j) = f_{i,j}\Big(g_{i,j,k}(A(i,k), B(k,j)) \text{ for } k \in S_{i,j}, K\Big)$$

with:

- $A = B = C$
- $g_{i,j,k}$ multiplies $L_{i,k} \cdot U_{k,j}$
- $f_{i,j}$ performs the sum, subtracts from $A_{i,j}$ (and divides by $U_{j,j}$ is $i > j$)
- I/O lower bound: $O(G/\sqrt{M}) = O(n^3/\sqrt{M})$
- Some algorithms attain this bound

Data Aware Algorithms – Part 1

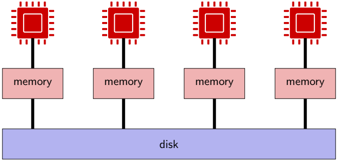
Pebble game models

Algorithm Design and Data Movement: the Matrix Product Case

Analysis and Lower Bounds for Parallel Algorithms

Conclusion

Matrix Multiplication Lower Bound for P processors



Lemma.

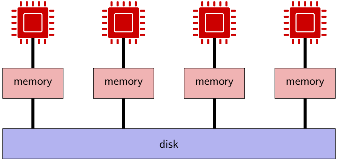
Consider a conventional matrix multiplication performed on P processors with distributed memory. A processor with memory M that perform W elementary products must send or receive at least $\frac{W}{2\sqrt{2}\sqrt{M}} - M$ elements.

Theorem.

Consider a conventional matrix multiplication on P processors, each with a memory M. Some processor has a volume of I/O at least $\frac{n^3}{2\sqrt{2}P\sqrt{M}} - M$.

NB: bound useful only when $M < n^2/(2P^{2/3})$

Matrix Multiplication Lower Bound for P processors



Lemma.

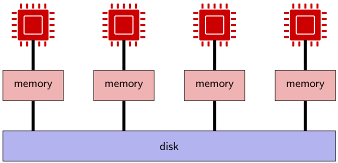
Consider a conventional matrix multiplication performed on P processors with distributed memory. A processor with memory M that perform W elementary products must send or receive at least $\frac{W}{2\sqrt{2}\sqrt{M}} - M$ elements.

Theorem.

Consider a conventional matrix multiplication on P processors, each with a memory M. Some processor has a volume of I/O at least $\frac{n^3}{2\sqrt{2}P\sqrt{M}} - M$.

NB: bound useful only when $M < n^2/(2P^{2/3})$

Matrix Multiplication Lower Bound for P processors



Lemma.

Consider a conventional matrix multiplication performed on P processors with distributed memory. A processor with memory M that perform W elementary products must send or receive at least $\frac{W}{2\sqrt{2}\sqrt{M}} - M$ elements.

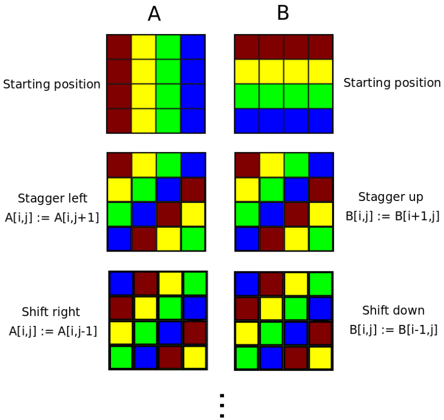
Theorem.

Consider a conventional matrix multiplication on P processors, each with a memory M. Some processor has a volume of I/O at least $\frac{n^3}{2\sqrt{2}P\sqrt{M}} - M$.

NB: bound useful only when $M < n^2/(2P^{2/3})$

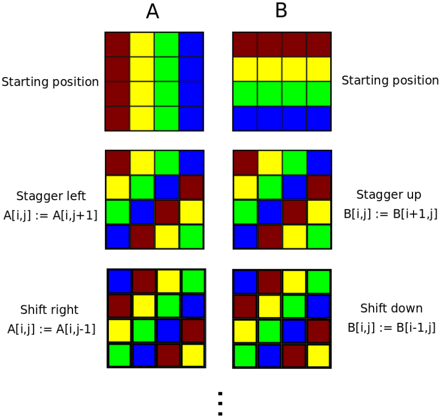
Cannon's 2D algorithm

- Processors organized on a square 2D grid of size $\sqrt{P} \times \sqrt{P}$
- A, B, C matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
  Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product
- First reallign matrices:
  - Shift $A_{i,j}$ blocks to the left by i (wraparound)
  - Shift $B_{i,j}$ blocks to the top by j (wraparound)
- After computation, shift A blocks right shift B blocks down
- Total I/O volume: ?
- Storage: ?



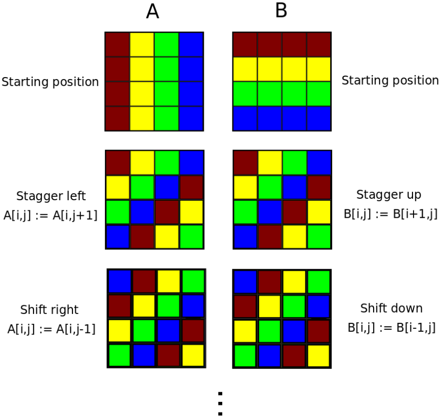
Cannon's 2D algorithm

- Processors organized on a square 2D grid of size $\sqrt{P} \times \sqrt{P}$
- A, B, C matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
  Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product
- First reallign matrices:
  - Shift $A_{i,j}$ blocks to the left by i (wraparound)
  - Shift $B_{i,j}$ blocks to the top by j (wraparound)
- After computation,
  shift A blocks right
  shift B blocks down
- Total I/O volume: ?
- Storage: ?



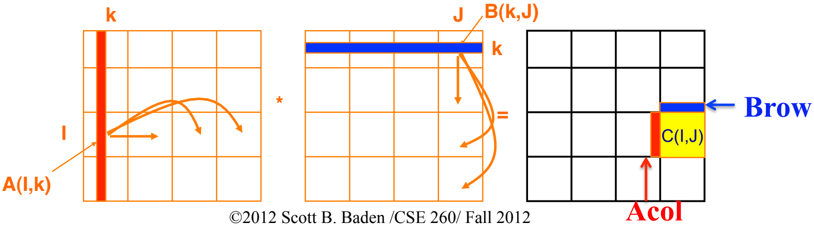
Cannon's 2D algorithm

- Processors organized on a square 2D grid of size $\sqrt{P} \times \sqrt{P}$
- A, B, C matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
  Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product
- First reallign matrices:
  - Shift $A_{i,j}$ blocks to the left by i (wraparound)
  - Shift $B_{i,j}$ blocks to the top by j (wraparound)
- After computation,
  shift A blocks right
  shift B blocks down
- Total I/O volume: $O(n^2\sqrt{P})$
- Storage: $O(n^2/P)$ per processor



Other 2D Algorithm: SUMMA

- SUMMA: Scalable Universal Matrix Multiplication Algorithm
- Same 2D grid distribution
- At each step k, column k of A and row k of B are broadcasted (from processors owning the data)
- Each processor computes a local contribution (outer-product)



- Smaller communications $\Rightarrow$ smaller temporary storage
- Same I/O volume: $O(n^2\sqrt{P})$

I/O Lower Bound for 2D algorithms

Theorem.

Consider a conventional matrix multiplication on P processors each with $O(n^2/P)$ storage, some processor has a I/O volume at least $\Theta(n^2/\sqrt{P})$.

Proof: Previous result: $O(n^3/P\sqrt{M})$ with $M = n^2/P$.

- When balanced, total I/O volume: $\Theta(n^2\sqrt{P})$
- Both Cannon's algorithm and SUMMA are optimal

Can we do better?

I/O Lower Bound for 2D algorithms

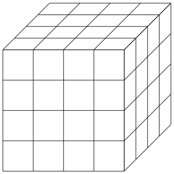
Theorem.
Consider a conventional matrix multiplication on P processors each with $O(n^2/P)$ storage, some processor has a I/O volume at least $\Theta(n^2/\sqrt{P})$.

Proof: Previous result: $O(n^3/P\sqrt{M})$ with $M = n^2/P$.

- When balanced, total I/O volume: $\Theta(n^2\sqrt{P})$
- Both Cannon's algorithm and SUMMA are optimal
  $\Rightarrow$ among 2D algorithms! (memory limited to $O(n^2/P)$)

Can we do better?

3D Algorithm

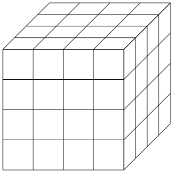


- Consider 3D grid of processor: $q \times q \times q$ ($q = P^{1/3}$)
- Processor i, j, k owns blocks $A_{i,k}, B_{k,j}, C_{i,j}^{(k)}$
- Matrices are replicated (including C)
- Each processor computes its local contribution
- Then summation of the various $C_{i,j}^{(k)}$ for all k
- Memory needed: ?
- Total I/O volume: ?

Lower Bound:

- Previous theorem does not give useful bound ($M = \Theta(n^2 P^{1/3})$)
- More complex analysis shows that the I/O volume on some processor is $\Theta(n^2/P^{2/3})$

3D Algorithm

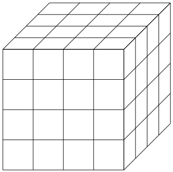


- Consider 3D grid of processor: $q \times q \times q$ ($q = P^{1/3}$)
- Processor i, j, k owns blocks $A_{i,k}, B_{k,j}, C_{i,j}^{(k)}$
- Matrices are replicated (including C)
- Each processor computes its local contribution
- Then summation of the various $C_{i,j}^{(k)}$ for all k
- Memory needed: $O(n^2/q^2) = O(n^2/P^{2/3})$ per processor
- Total I/O volume: $O(n^2/q^2 \times q^3) = O(n^2 q) = O(n^2 P^{1/3})$

Lower Bound:

- Previous theorem does not give useful bound ($M = \Theta(n^2 P^{1/3})$)
- More complex analysis shows that the I/O volume on some processor is $\Theta(n^2/P^{2/3})$

3D Algorithm



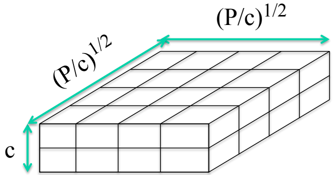
- Consider 3D grid of processor: $q \times q \times q$ ($q = P^{1/3}$)
- Processor i, j, k owns blocks $A_{i,k}, B_{k,j}, C_{i,j}^{(k)}$
- Matrices are replicated (including C)
- Each processor computes its local contribution
- Then summation of the various $C_{i,j}^{(k)}$ for all k
- Memory needed: $O(n^2/q^2) = O(n^2/P^{2/3})$ per processor
- Total I/O volume: $O(n^2/q^2 \times q^3) = O(n^2 q) = O(n^2 P^{1/3})$

Lower Bound:

- Previous theorem does not give useful bound ($M = \Theta(n^2 P^{1/3})$)
- More complex analysis shows that the I/O volume on some processor is $\Theta(n^2/P^{2/3})$

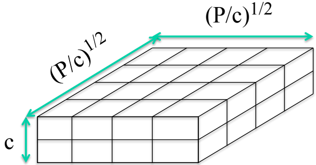
2.5D Algorithm (1/2)

- 3D algorithm requires large memory on each processor ($P^{1/3}$ copies of each matrices)
- What if we have space for only $1 < c < P^{1/3}$ copies ?
- Assume each processor has a memory $M = O(cn^2/P)$
- Arrange processors in $\sqrt{P/c} \times \sqrt{P/c} \times c$ grid:
  c layers, each layer with P/c processors in square grid
- A, B, C
  distributed by blocks of size $n\sqrt{c/P} \times n\sqrt{c/P}$, replicated on each layer



- NB: $c = 1$ gets 2D, $c = P^{1/3}$ gives 3D

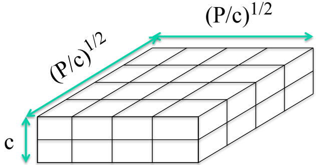
2.5D Algorithm (2/2)



- Each layer responsible for a fraction $1/c$ of Cannon's alg.: Different initial shifts of A and B
- Finally, sum C over layers
- Total I/O volume: $O(n^2/\sqrt{P/c})$
  - Replication, initial shift, final sum: $O(n^2 c)$
  - c layers of fraction $1/c$ of Cannon's alg. with grid size $\sqrt{P/c}$: $O\left(n^2\sqrt{P/c}\right)$
- Reaches lower bound on I/Os per processor:

$$O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O(n^2/\sqrt{cP})$$

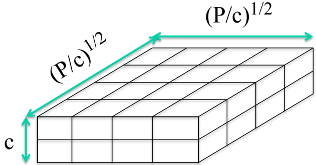
2.5D Algorithm (2/2)



- Each layer responsible for a fraction $1/c$ of Cannon's alg.: Different initial shifts of A and B
- Finally, sum C over layers
- Total I/O volume: $O(n^2/\sqrt{P/c})$
  - Replication, initial shift, final sum: $O(n^2 c)$
  - c layers of fraction $1/c$ of Cannon's alg. with grid size $\sqrt{P/c}$: $O\left(n^2\sqrt{P/c}\right)$
- Reaches lower bound on I/Os per processor:

$$O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O(n^2/\sqrt{cP})$$

2.5D Algorithm (2/2)

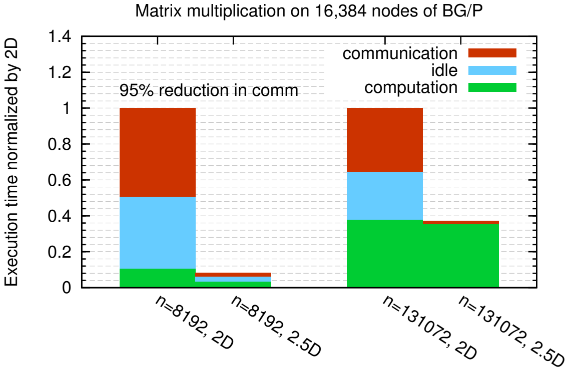


- Each layer responsible for a fraction $1/c$ of Cannon's alg.: Different initial shifts of A and B
- Finally, sum C over layers
- Total I/O volume: $O(n^2/\sqrt{P/c})$
  - Replication, initial shift, final sum: $O(n^2 c)$
  - c layers of fraction $1/c$ of Cannon's alg. with grid size $\sqrt{P/c}$: $O\left(n^2\sqrt{P/c}\right)$
- Reaches lower bound on I/Os per processor:

$$O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O(n^2/\sqrt{cP})$$

Performance on Blue Gene P

C=16



Source Jim Demmel

Data Aware Algorithms – Part 1

Take-aways

- Data movements (I/Os and communication between processes) have a large impact on the efficiency of algorithms
- Different algorithms with different computational complexity may exhibit very different I/O behaviors
- We can prove lower bound on the amount of I/O or communications for specific operations
- I/O (asymptotically) optimal algorithms for linear algebra operations
- Communication-avoiding algorithms for parallel processing

See you tomorrow!