Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models

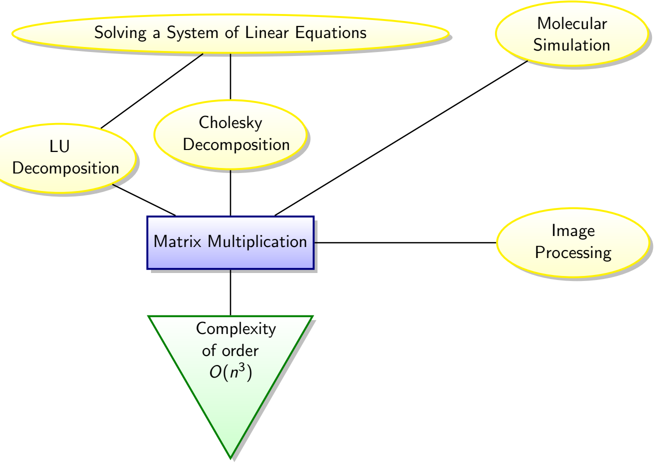
David Clarke   Alexey Lastovetsky   Vladimir Rychkov

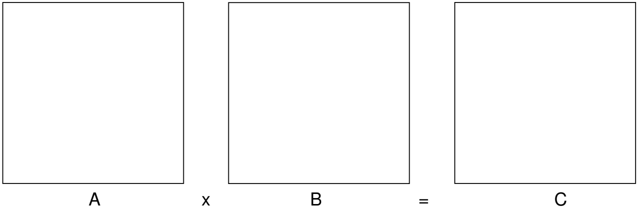
Heterogeneous Computing Laboratory, School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 4, Ireland
http://hcl.ucd.ie

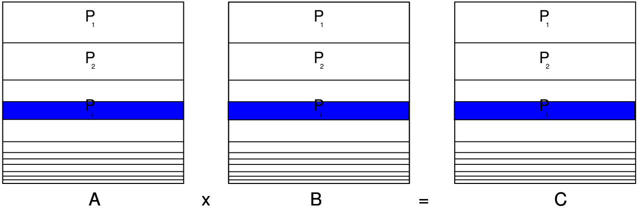
HeteroPar'2011

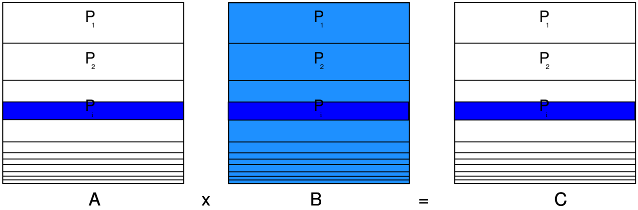
Outline

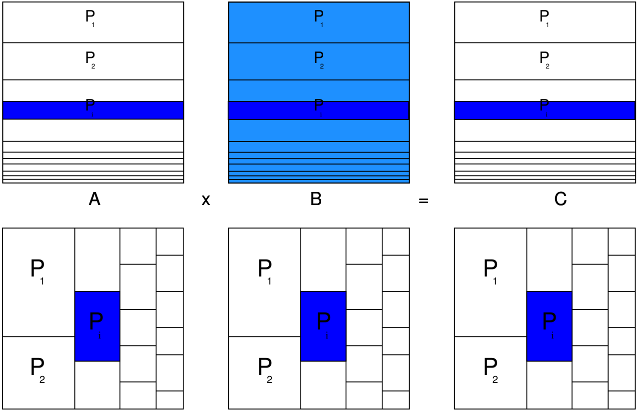
Why optimize Matrix Multiplication?

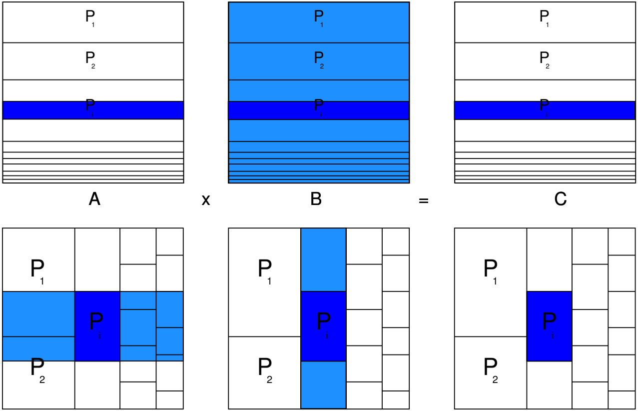



A
x
B
=
C


P
1
P
2
P
i
A
x
B
=
C

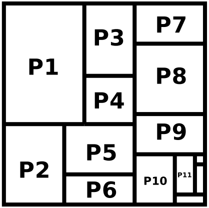

P1
P2
Pi
A
x
B
=
C


P1
P2
Pi
A
x
B
=
C
P1
P2
Pi

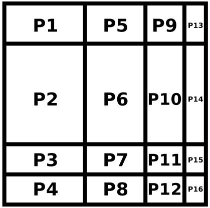

P 1
P 2
P i
A
x
B
=
C

Optimising Parallel Matrix Multiplication on a Heterogeneous Platform

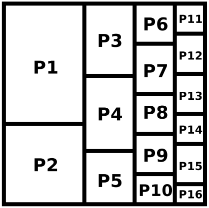
- Partition in proportion to processor speed.
- Minimise volume of communication.
- Partition in proportion to interconnect speed.



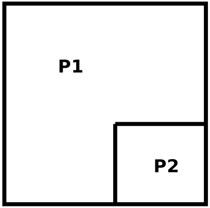
General (NP complete)



Cartesian

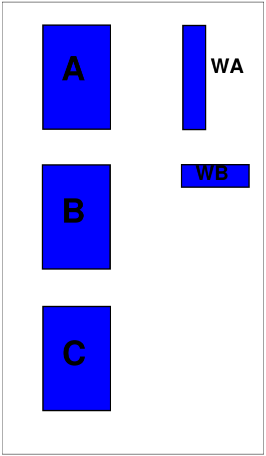


Column-Based

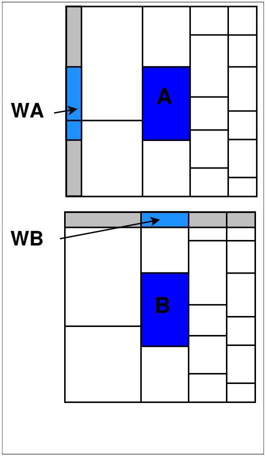


Square-Corner

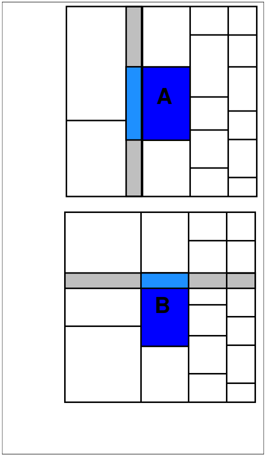
allocate and initialise matrices A, B, C;
allocate workspace WA, WB;
for $k = 0 \rightarrow N - 1$ **do**
  if (is pivot row) **then**
    point WB to local pivot row of B;
    Broadcast WB to all in column;
  else
    Receive WB;
  end if
  if (is pivot column) **then**
    point WA to local pivot column of A;
    Send WA horizontally;
  else
    receive WA;
  end if
  DGEMM($\ldots$, WA, WB, C, $\ldots$);
end for



allocate and initialise matrices A, B, C;
allocate workspace WA, WB;
for $k = 0 \to N - 1$ **do**
  if (is pivot row) **then**
    point WB to local pivot row of B;
    Broadcast WB to all in column;
  else
    Receive WB;
  end if
  if (is pivot column) **then**
    point WA to local pivot column of A;
    Send WA horizontally;
  else
    receive WA;
  end if
  DGEMM(…, WA, WB, C, …);
end for



allocate and initialise matrices A, B, C;
allocate workspace WA, WB;
for $k = 0 \rightarrow N - 1$ **do**
  if (is pivot row) **then**
    point WB to local pivot row of B;
    Broadcast WB to all in column;
  else
    Receive WB;
  end if
  if (is pivot column) **then**
    point WA to local pivot column of A;
    Send WA horizontally;
  else
    receive WA;
  end if
  DGEMM($\ldots$, WA, WB, C, $\ldots$);
end for



Benchmarking on each processor must be independent of other processors: serial code.

allocate and initialise matrices A, B, C;
allocate workspace WA, WB;

start timer;
MPI_Send(A, ..., MPI_COMM_SELF);
MPI_Recv(WA, ..., MPI_COMM_SELF);
memcpy(WB, B, ...);
DGEMM(..., WA, WB, C, ...);
stop timer;

free memory;

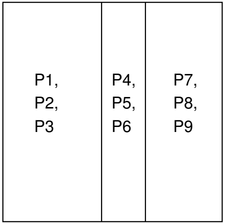
Matrix Partitioning Algorithms

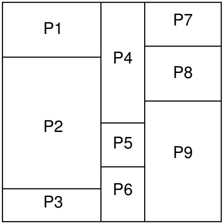
- Column-Based Partitioning
  (Kalinov & Lastovetsky 1999) (**KL**)
- Minimising Total Communication Volume
  (Beaumont, Boudet, Rastello, Robert, 2001) (**BR**)
- 1D Functional Performance Model-based Partitioning
  (Lastovetsky, Reddy, 2007) (**FPM1D**)
- 2D Functional Performance Model-based Partitioning
  (Lastovetsky, Reddy, 2010) (**FPM-KL**)

- New Two-Dimensional Matrix Partitioning Algorithm
  (**FPM-BR**)

Column-Based Partitioning (**KL**)

- Processors are arranged into columns.
- The width of each column is in proportion to the sum of the speeds of the processors in that column.
- Within each column the heights are calculated in proportion to speed.





Column-Based Partitioning (**KL**)

- Processors are arranged into columns.
- The width of each column is in proportion to the sum of the speeds of the processors in that column.
- Within each column the heights are calculated in proportion to speed.

- However, communication cost is not taken into account.
- Uses inaccurate, single-value performance model of processor speed.

Minimising Total Communication Volume (**BR**)

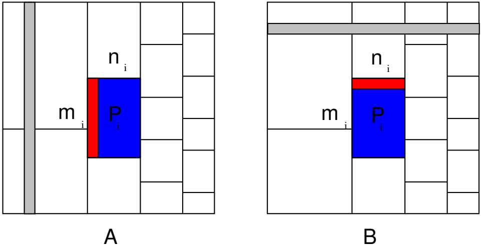
- Column-based algorithm.
- Computes:
  - Optimum number of columns
  - Optimum number of processors in each column
- Such that:
  - Workload is distributed in proportion to speed,
  - Total volume of communication is minimised.

Minimising Total Communication Volume (**BR**)

- Column-based algorithm.
- Computes:
  - Optimum number of columns
  - Optimum number of processors in each column
- Such that:
  - Workload is distributed in proportion to speed,
  - Total volume of communication is minimised.

- However, uses inaccurate, single-value performance model of processor speed.

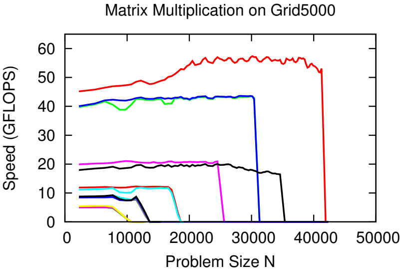
Minimising Total Communication Volume (**BR**)



Total volume of communication $= \sum_i^p (m_i + n_i)$
"the sum of the half perimeters"
minimised when $m_i \approx n_i$

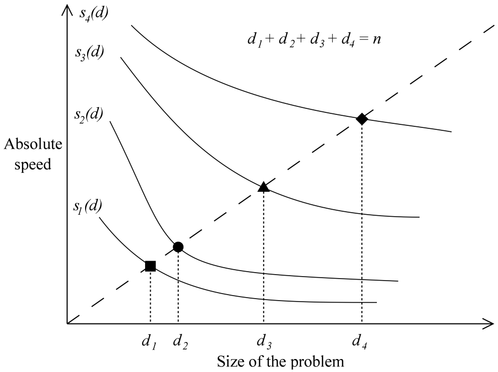
Realistic Performance Models

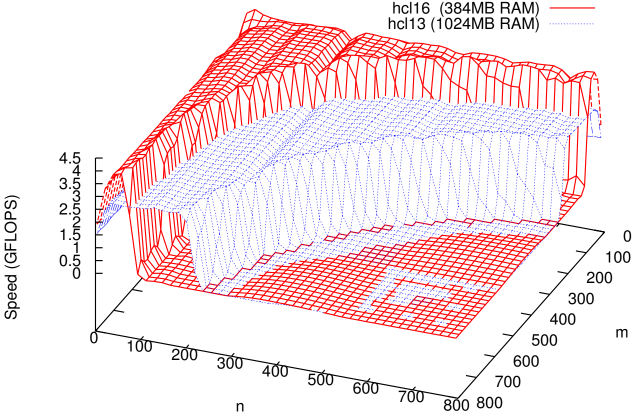
- Traditionally, processor performance is defined by a constant number.
- In reality, speed is a function of problem size.
- Algorithms based on constant performance models are only applicable for limited problem sizes.



1D Functional Performance Model-based Partitioning (**FPM1D**)

- Problem is solved geometrically by noting that the points $(d_i, s_i(d_i))$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.



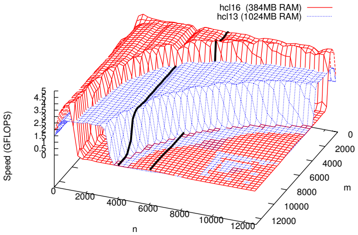

hcl16 (384MB RAM)
hcl13 (1024MB RAM)
Speed (GFLOPS)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
n
0
100
200
300
400
500
600
700
800
m
0
100
200
300
400
500
600
700
800

2D Functional Performance Model-based Partitioning (**FPM-KL**)

- Column-based partitioning with 2D performance models.
- Processors are arranged in a grid $p \times q$
- Column widths are initially distributed $n_j = N/q \ \forall j$.

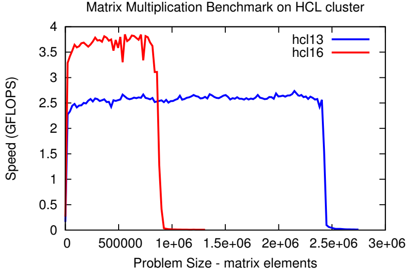
Iterating:

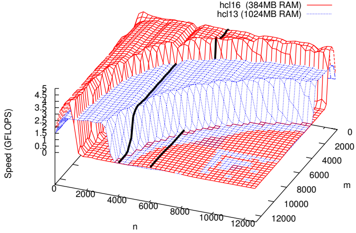
1. 1D models are sliced from 2D at column widths.
2. Optimum partitioning within each column is solved with **FPM1D** algorithm.
3. If disbalance $< \epsilon$ then finished, else continue.
4. Single value speeds from this partitioning used to calculate new column widths.



2D Functional Performance Model-based Partitioning (**FPM-KL**)

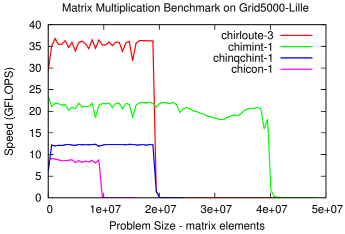
- Does not take communication cost into account.
- Processor grid is fixed.
- Relies on single speed values to calculate new column widths.
- Building full 2D models is expensive

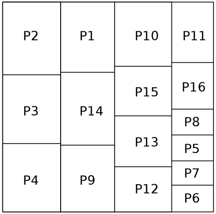




New Two-Dimensional Matrix Partitioning Algorithm (**FPM-BR**)

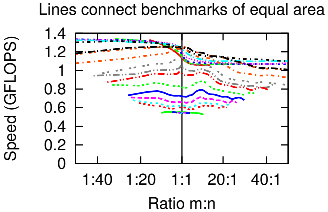
- Height m_i and width n_i combined into one parameter, area $d_i = m_i \times n_i$.
- Square areas are benchmarked $m = n = \sqrt{d}$.
- Partition with **FPM1D** algorithm, find area rectangles.
- BR algorithm computes ordering and shape of these rectangles.

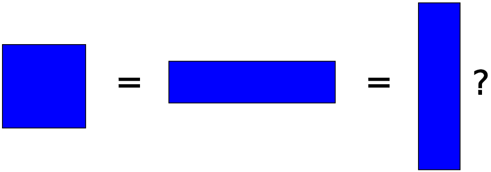




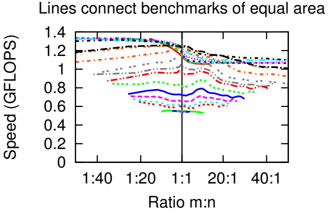
Assumption square area performance is the same as performance of any rectangle of the same area, $s(x,x) = s(x/c, c.x)$.

- not always true.

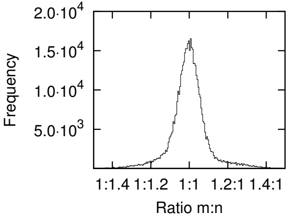


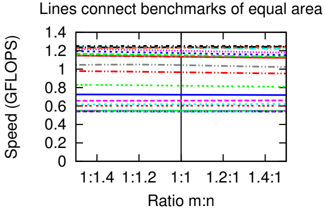


Assumption square area performance is the same as performance of any rectangle of the same area, $s(x, x) = s(x/c, c.x)$.
- not always true.

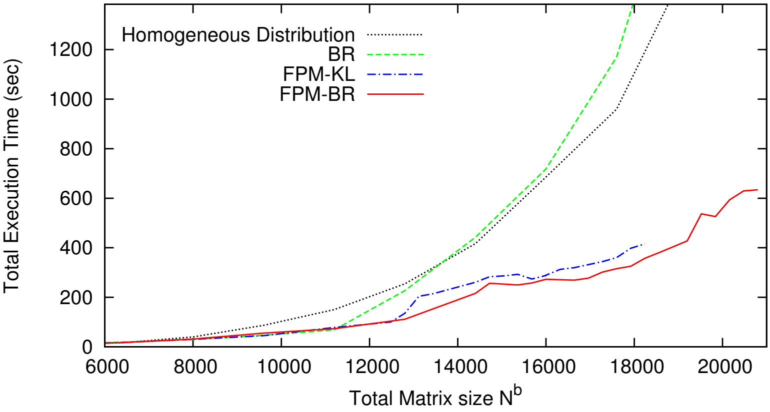


However, goal of **BR** algorithm is to make rectangles square.

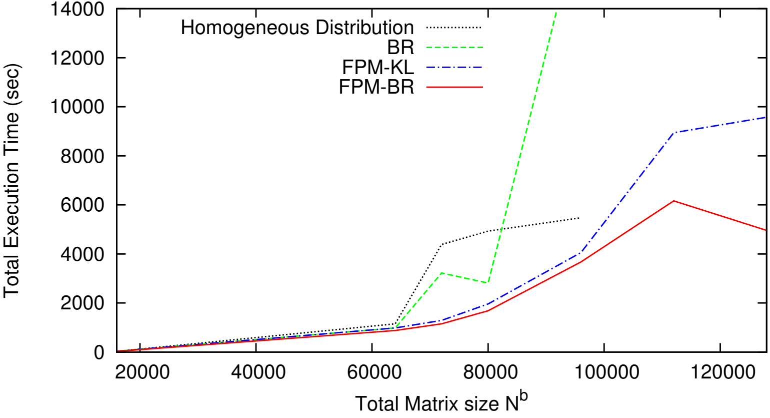




Experimental Results



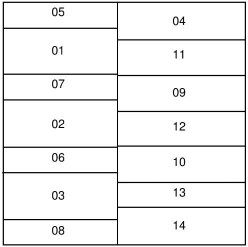
16 heterogeneous nodes, local HCL cluster.



64 nodes from Grid5000 Lille site (4 types of nodes).

Matrix partitioning for 14 nodes

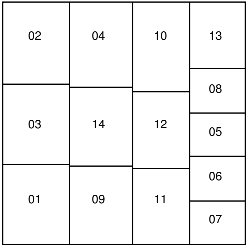
FPM-KL



TVC: 9

Time: 192.2sec

FPM-BR



TVC: 7.457

Time: 166.0sec

Conclusions

- New **FPM-BR** algorithm can outperform existing algorithms.
- Allows use of simpler 1D models.
- Total volume of communication is minimised.

Questions?