Using Multidimensional Solvers for Optimal Data Partitioning on Dedicated Heterogeneous HPC Platforms

Vladimir Rychkov Alexey Lastovetsky David Clarke

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

PaCT'2011

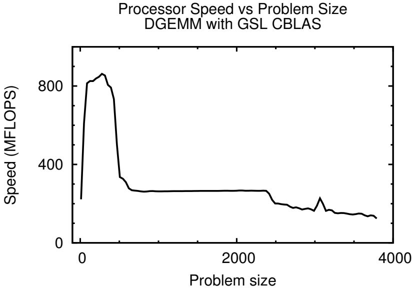
Outline

- Data-intensive parallel computational routines: computational workload is divisible and proportional to data size
  number of processors: p
  data partition: $n = d_1 + d_2 + \cdots + d_p$
- Dedicated heterogeneous HPC platforms: load is balanced when the execution times are equal:
  $t_1 = t_2 = \cdots = t_p$

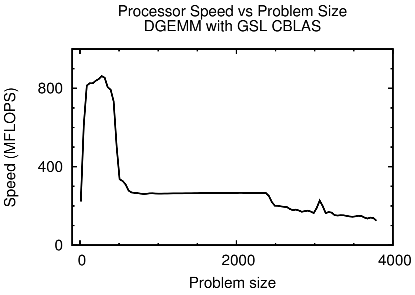
- Data-intensive parallel computational routines: computational workload is divisible and proportional to data size
  number of processors: p
  data partition: $n = d_1 + d_2 + \cdots + d_p$
- Dedicated heterogeneous HPC platforms: load is balanced when the execution times are equal:
  $t_1 = t_2 = \cdots = t_p$
- Processor speed: $s_i = \dfrac{d_i}{t_i}$
- Data partitioning problem:

$$\begin{cases} \dfrac{d_1}{s_1} = \dfrac{d_2}{s_2} = \cdots = \dfrac{d_p}{s_p} \\ d_1 + d_2 + \cdots + d_p = n \end{cases} \tag{1}$$

- Traditionally, processor performance is defined by a constant number: $s = const$
- In reality, speed is a function of problem size: $s = s(x)$

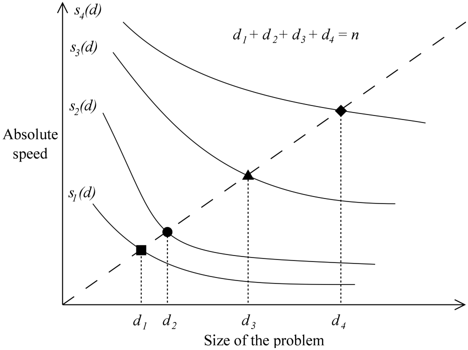


- Traditionally, processor performance is defined by a constant number: $s = const$
- In reality, speed is a function of problem size: $s = s(x)$
- Partitioning algorithms based on constant performance models are only applicable for limited problem sizes
- How to solve (1) with speed functions?

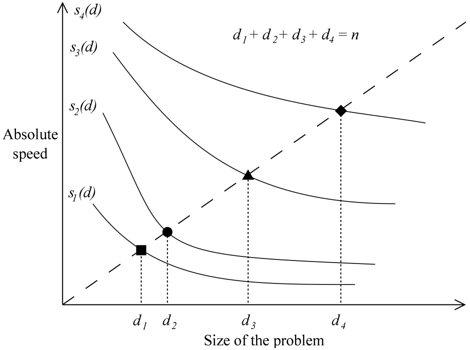


Layout

Geometrical solution: points $(d_i, s_i(d_i))$ on a line passing through the origin $\Rightarrow \dfrac{d_i}{s_i(d_i)} = const$ and $\sum d_i = n$

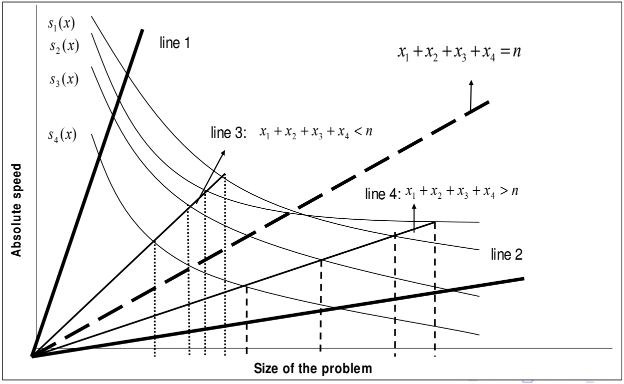


Geometrical solution: points $(d_i, s_i(d_i))$ on a line passing through the origin $\Rightarrow \dfrac{d_i}{s_i(d_i)} = const$ and $\sum d_i = n$



(2) Assumption: any straight line passing through the origin intersects speed functions only once

- The space of solutions: all lines passing through the origin
- Initial bounds for some $n^U < n$ and $n^L > n$:
  $x_1^U, \dots, x_p^U$: $\sum x_i^U = n^U$, and $x_1^L, \dots, x_p^L$: $\sum x_i^L = n^L$
- The region between two lines is iteratively bisected and the bounds are updated



Layout

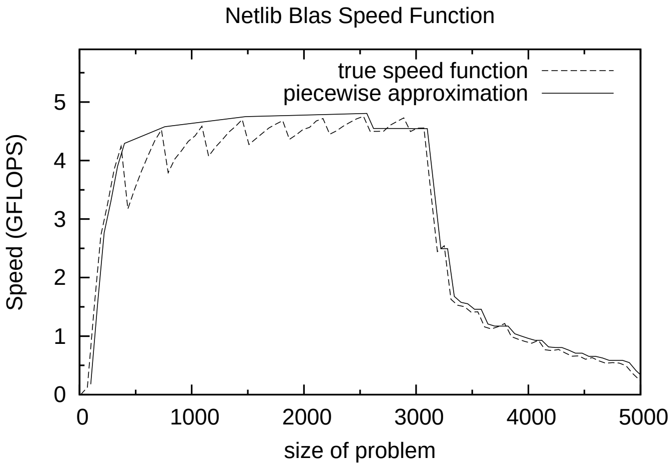
Restrictions on the shape of a speed function $s(x)$ to satisfy the assumption (2):

- Increasing and convex on $[0, X]$
- Decreasing on $[X, \infty]$

Fixes to the piecewise linear interpolation $\bar{s}(x)$ after adding a new data point (d^j, s^j):

- if $d^j \in [0, X]$, ensure that $s^{j-1} \leq s^j \leq s^{j+1}$ and $\dfrac{s^{j-1} - s^{j-2}}{d^{j-1} - d^{j-2}} \geq \dfrac{s^j - s^{j-1}}{d^j - d^{j-1}} \geq \dfrac{s^{j+1} - s^j}{d^{j+1} - d^j}$
- if $d^j \in [X, \infty]$, ensure that $s^{j-1} \geq s^j \geq s^{j+1}$

Result: inaccurate approximation of speed function



Layout

- Speeds are approximated by continuous differentiable functions of arbitrary shape

- Speeds are approximated by continuous differentiable functions of arbitrary shape
- Data partitioning problem (1) can be formulated as **multidimensional root finding** for the system of nonlinear equations $f(\mathbf{x}) = 0$, where

$$f(\mathbf{x}) = \begin{cases} n - \sum_{i=1}^{p} x_i \\ \frac{x_i}{s_i(x_i)} - \frac{x_1}{s_1(x_1)} & 2 \leq i \leq p \end{cases} \tag{3}$$

  $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$ represents data partition
- Optimal data partition is obtained after rounding of the root $\mathbf{x}^* = (x_1^*, \ldots, x_p^*)$ and distribution of the remainders

- Problem (3) can be solved by the Newton-Raphson method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - J(\mathbf{x}^k) f(\mathbf{x}^k) \tag{4}$$

- Initial guess: the equal data distribution

$$\mathbf{x}^0 = (n/p, \dots, n/p) \tag{5}$$

- Jacobian $J(\mathbf{x})$: (6)

$$J(\mathbf{x}) = \begin{pmatrix} -1 & -1 & \dots & -1 \\ -\frac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & \frac{s_2(x_2) - x_2 s_2'(x_2)}{s_2^2(x_2)} & 0 & 0 \\ \dots & 0 & \dots & 0 \\ -\frac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & 0 & 0 & \frac{s_p(x_p) - x_p s_p'(x_p)}{s_p^2(x_p)} \end{pmatrix}$$

To solve (4)-(6), we use the HYBRJ algorithm, a modified version of Powell's Hybrid method, implemented in the MINPACK library:

- retains the fast convergence of the Newton method
- reduces the residual when the Newton method is unreliable
- requires differentiable speed functions

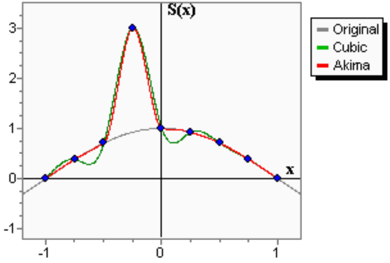
Layout

Approximations of speed function

- Piecewise linear interpolation: undefined derivative at breakpoints
- Splines of higher orders: differentiable but may yield significant oscillations

Approximations of speed function

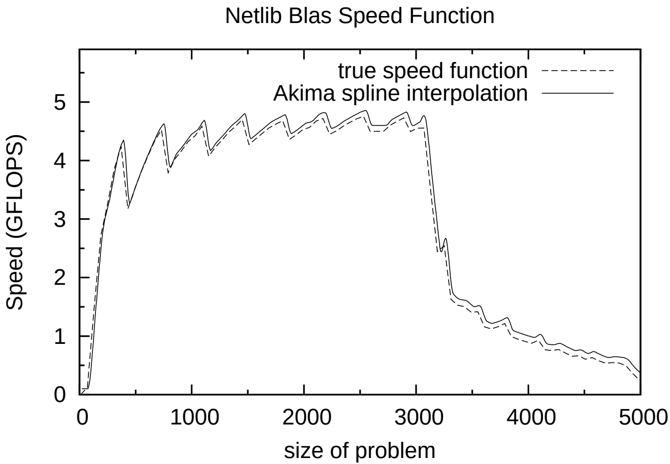
- Piecewise linear interpolation: undefined derivative at breakpoints
- Splines of higher orders: differentiable but may yield significant oscillations
- **Akima spline interpolation**: non-linear but stable to outliers



Akima spline interpolation

- Built from piecewise third order polynomials
- Computationally efficient:
  - no need to solve large equation systems
  - a small number of the neighbour points is taken into account
- Interpolation error in the inner area $O(h^2)$

Result: accurate approximation of speed function



Layout

Parallel Computational Iterative Routine

$$\mathbf{x}^{k+1} = f(\mathbf{x}^k) \qquad \mathbf{x}^k \in \mathbb{R}^n \qquad f : \mathbb{R}^n \to \mathbb{R}^n$$

- Data is partitioned over all processors: $n = d_1^k + d_2^k + \cdots + d_p^k$
- Some independent calculations are carried out in parallel
- Some data synchronisation takes place

Parallel Computational Iterative Routine

$$\mathbf{x}^{k+1} = f(\mathbf{x}^k) \qquad \mathbf{x}^k \in \mathbb{R}^n \qquad f : \mathbb{R}^n \to \mathbb{R}^n$$

- Data is partitioned over all processors: $n = d_1^k + d_2^k + \cdots + d_p^k$
- Some independent calculations are carried out in parallel
- Some data synchronisation takes place

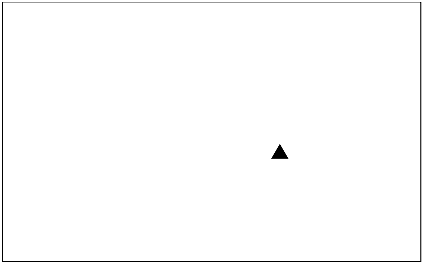
Model-based Dynamic Load Balancing

- At each iteration, execution times measured and sent to root
- Approximations of speed functions $\bar{s}_i(x)$ updated by adding the point $(d_i^k, d_i^k/t_i^k)$
- If relative difference between times $> \epsilon$, data partitioning algorithm calculates new data partition $\mathbf{d}^{k+1}$
- $\mathbf{d}^{k+1}$ broadcasted to all processors; data redistributed

Layout

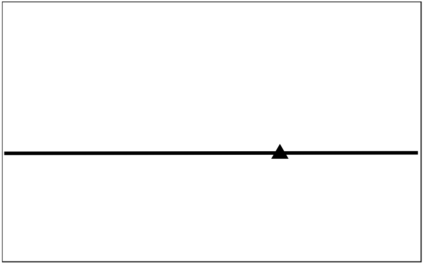
First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \frac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$



First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

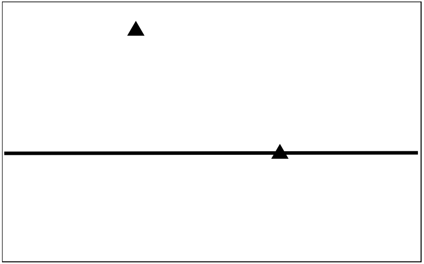


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

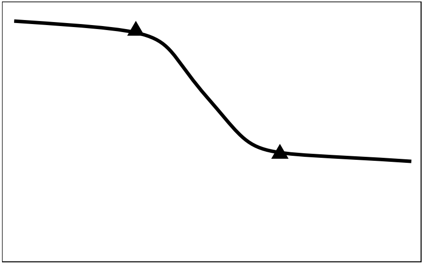


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

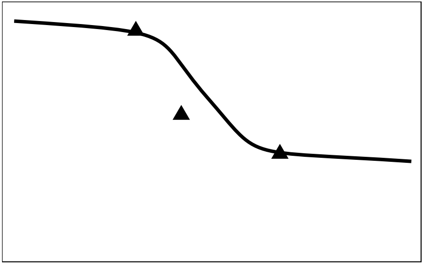


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

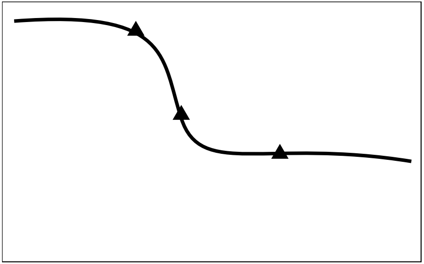


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

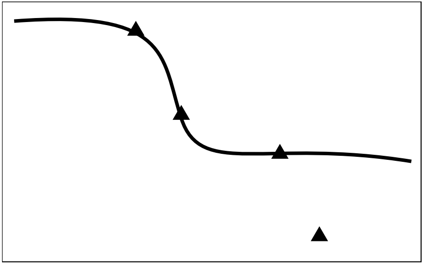


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

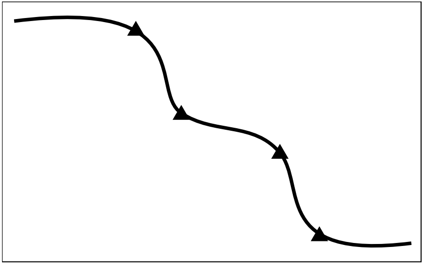


First iteration Point $(n/p, s_i^0)$ with speed $s_i^0 = \dfrac{n/p}{t_i(n/p)}$

First function approximation $\bar{s}_i(x) \equiv s_i^0$

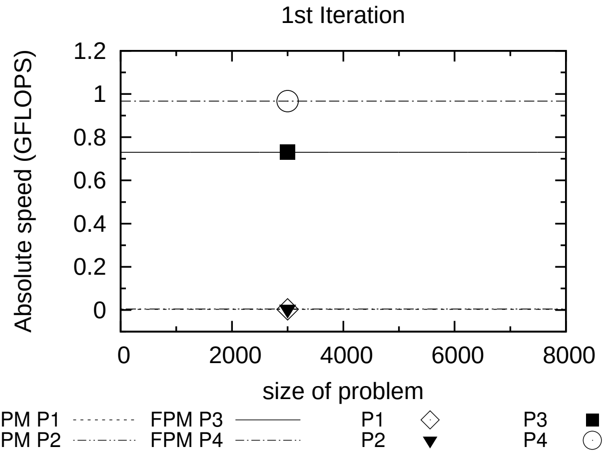
Subsequent iterations Point (d_i^k, s_i^k) with speed $s_i^k = \dfrac{d_i^k}{t_i(d_i^k)}$

Approximation $\bar{s}_i(x)$ updated by adding the point

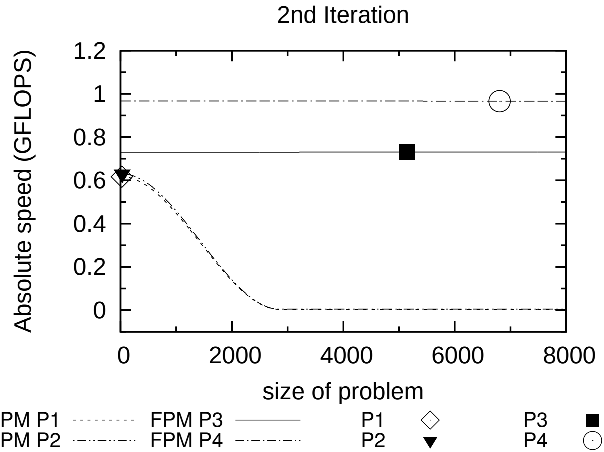


Layout

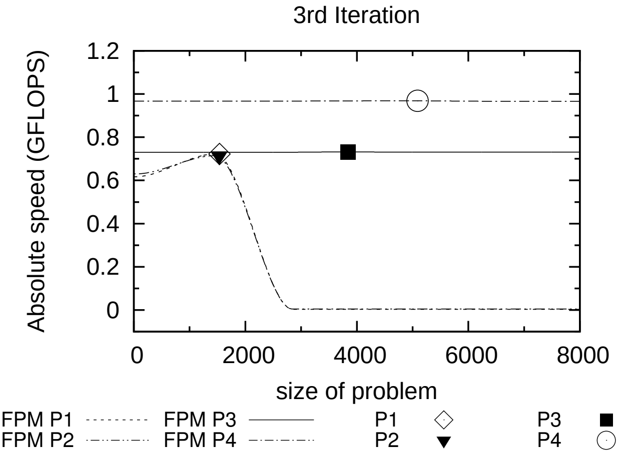
Experimental Results: Jacobi Method



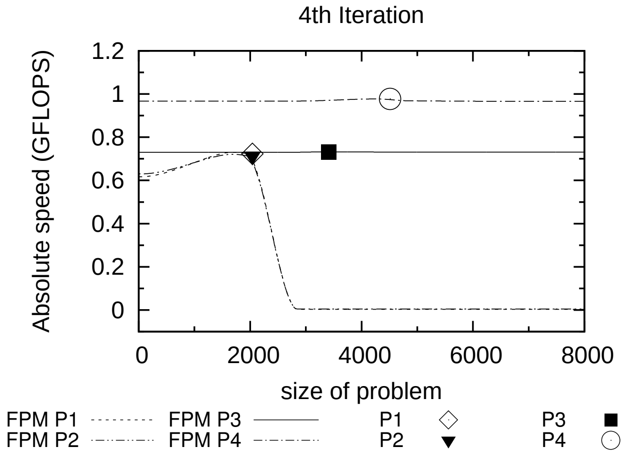
Experimental Results: Jacobi Method



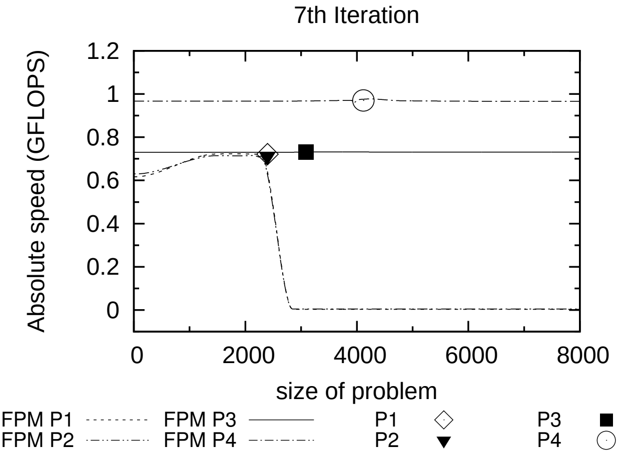
Experimental Results: Jacobi Method



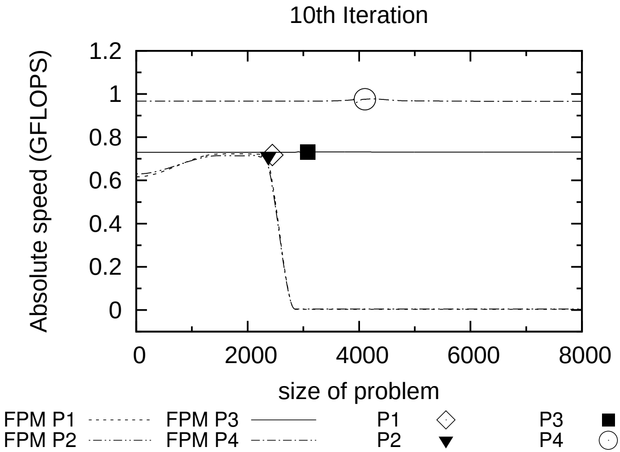
Experimental Results: Jacobi Method



Experimental Results: Jacobi Method



Experimental Results: Jacobi Method



Conclusions

- Traditional data partitioning algorithms only work for problems which fit into the main memory of all processors.
- The proposed algorithm, based on accurate functional performance models, can balance for all problem sizes.
- No prior information about the heterogeneity and memory hierarchy of the platform needed as inputs into the algorithm.
- Can be deployed self adaptively on any dedicated platform.

Project web page: `http://hcl.ucd.ie/project/fupermod`

Heterogeneous Computing Laboratory

School of Computer Science and Informatics
University College Dublin

Science Foundation Ireland