university of groningen

Supercomputing for Giga- and Tera-Pixel Image Scales, Part III: Multi-Scale and Distributed Connected Filtering

Michael H. F. Wilkinson

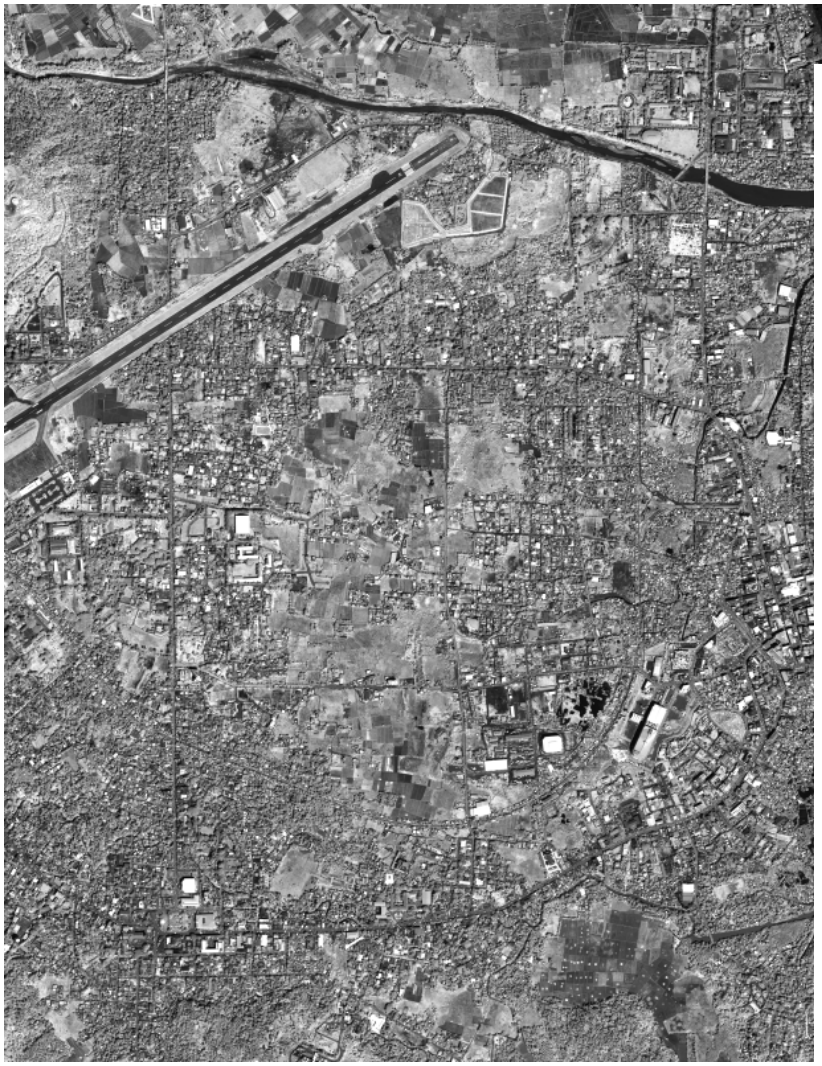
Johann Bernoulli Institute
University of Groningen

Overview

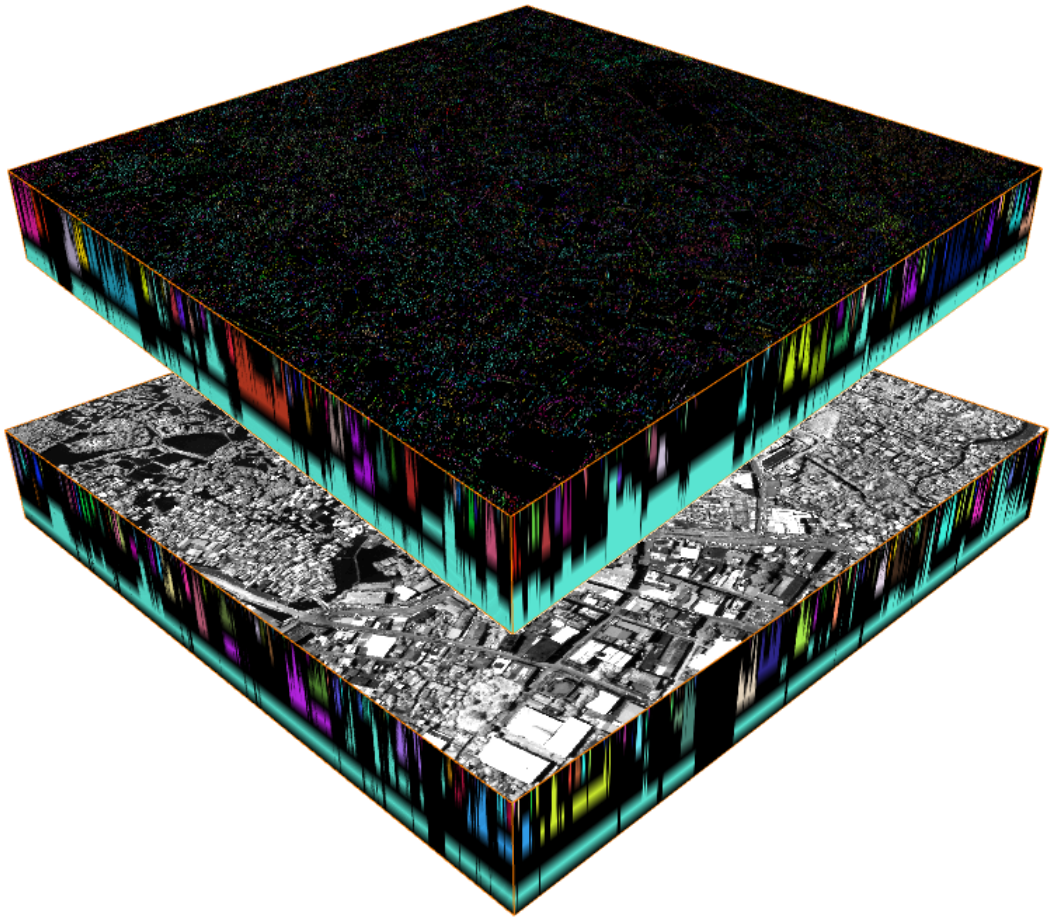
- Multi-Scale Analysis by Component Trees
  - Differential Morphological Profiles
  - Differential Area Profiles
  - Algorithms
- Distributed Computation of Connected Filters
  - Modified Max-tree Forests
  - Boundary Trees
  - Algorithms

Quickbird image

Morphological Profiles



Quickbird image



Differential Morphological Profile

High-performance **I**mage **P**rocessing **A**lgorithms for **R**emote **S**ensing

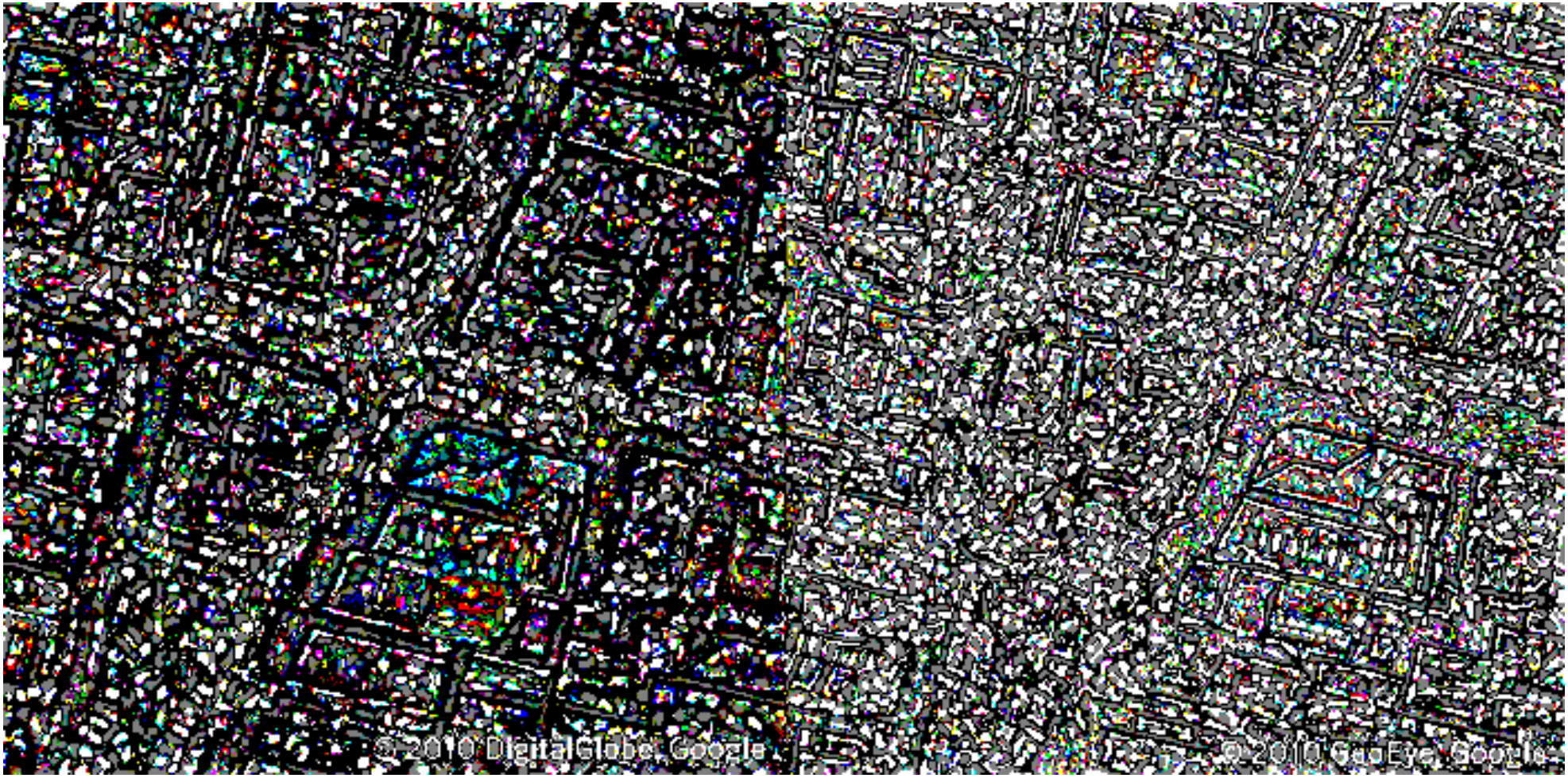
- Georgios Ouzounis
- Martino Pesaresi
- Pierre Soille

JRC EUROPEAN COMMISSION

- Michael Wilkinson

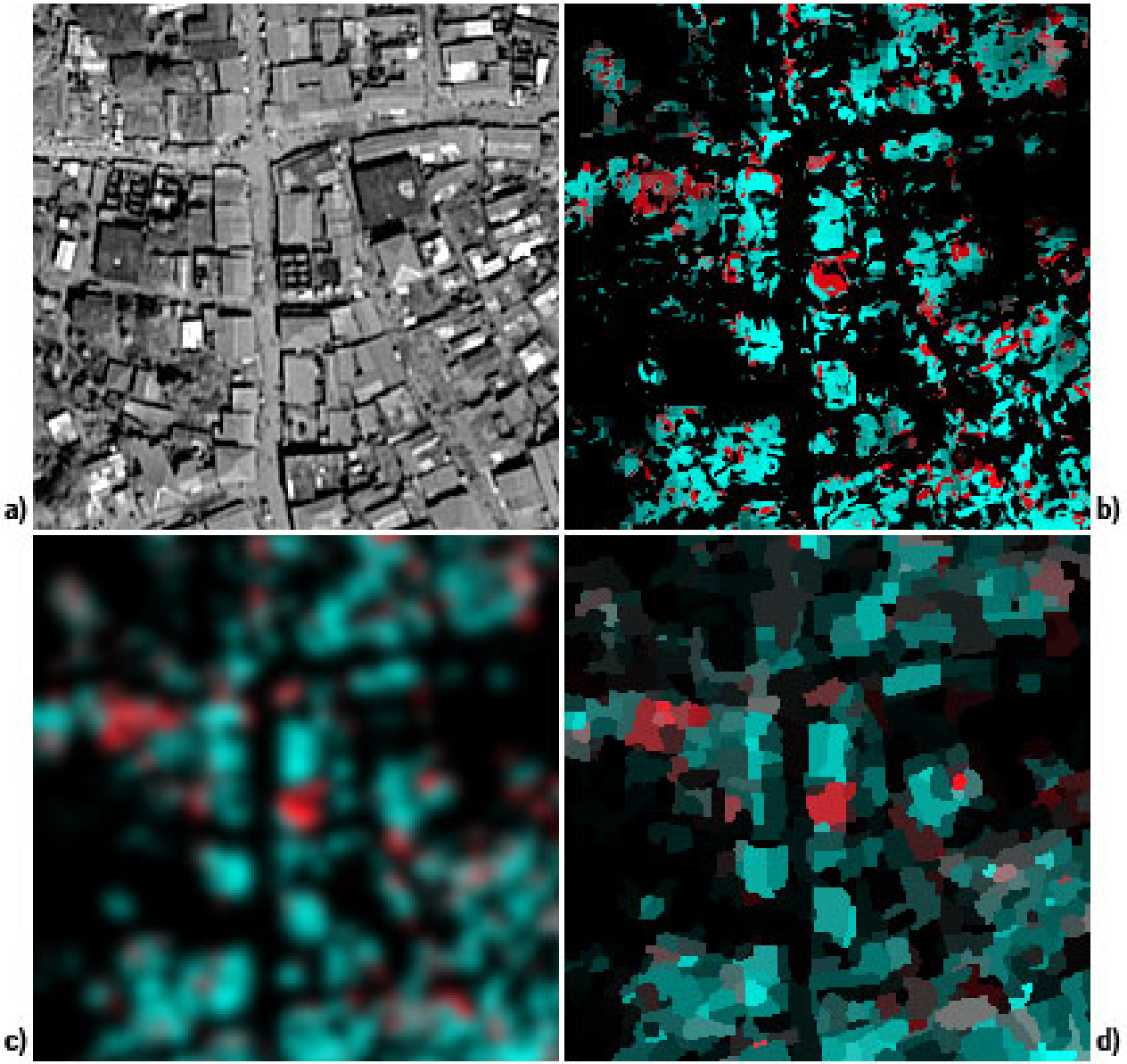
university of groningen

Use-case Port-au-Prince



before

after


a)
b)
c)
d)

Scale of the Problem

Data set	Classical Algorithm	Improved Algorithm
Haiti 2010	34,000 years	104 days
Tsunami 2004	500,000 years	2.5 years

Scale of the Problem

Data set	Classical Algorithm	Improved Algorithm
Haiti 2010	34,000 years	104 days
Tsunami 2004	500,000 years	2.5 years

This is useless!

Scale of the Problem

Data set	Classical Algorithm	Improved Algorithm
Haiti 2010	34,000 years	104 days
Tsunami 2004	500,000 years	2.5 years

This is useless!

Haiti can now be done in

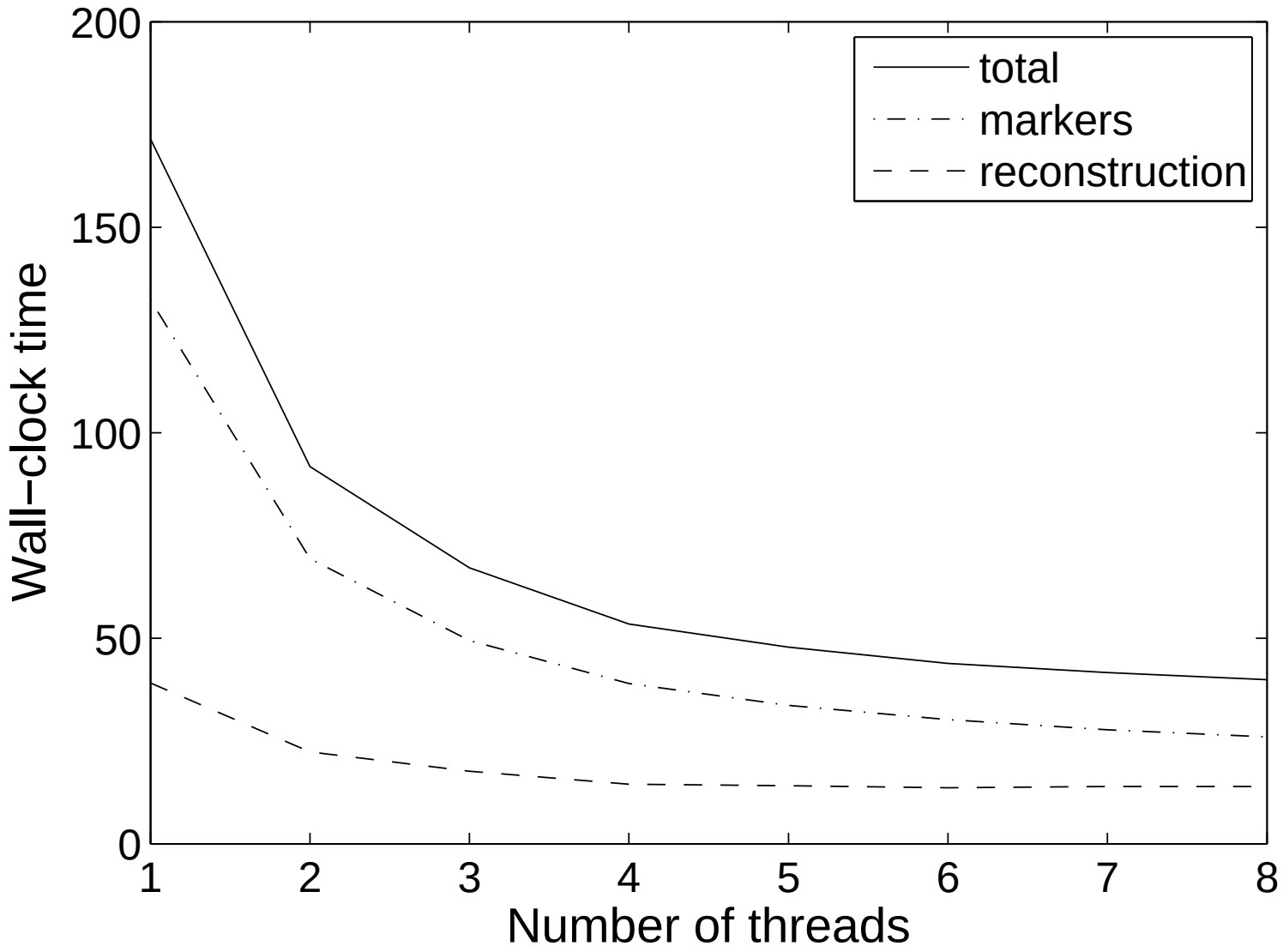
1 h 22 min

Basic Algorithm

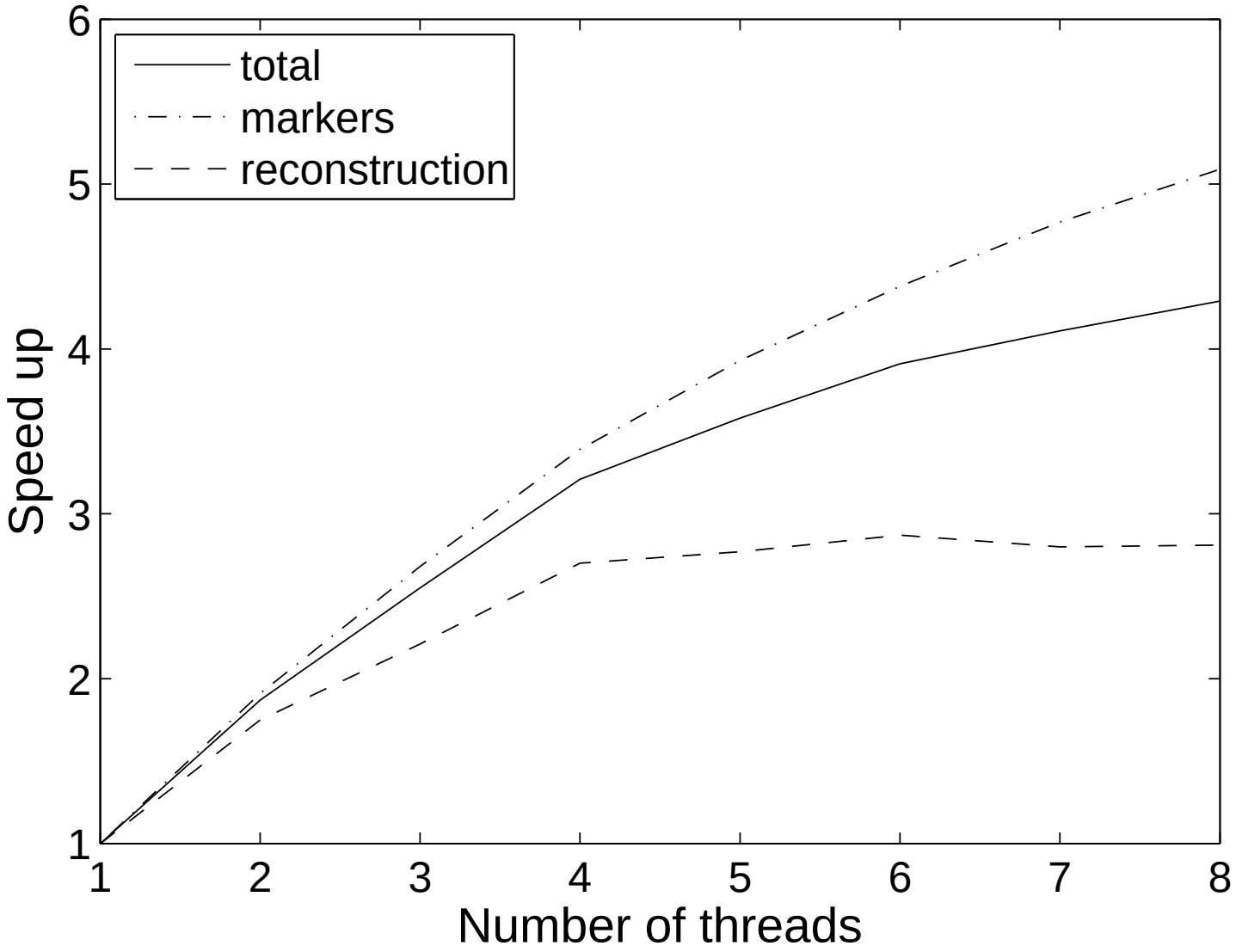
- Compute all markers using parallel erosions and store each scale i in an image $out[i]$;
- Compute a single Max-Tree in parallel, maintaining the maximum marker for all scales i for each node n in $out[i][n]$;
- For all scales i
  - Set all nodes to invalid indicating that they have not been filtered yet (in parallel);
  - Compute reconstruction (in parallel) for scale i, storing the result in $out[i]$;
- Compute differences between scales (in parallel);
- Output respective instance of DMP vector field.

procedure *MaxTreeReconstruct* (*Vp* : *Section*; **var** $node$: *Max-Tree*; **var** out : *Image*)
  for all $v \in Vp$ **do**
    if not $node[v].valid$ **then**
      $w := v$;
      while $Par(w) \neq \perp \wedge$ **not** $node[w].valid \wedge f[Par(w)] > out[Par(w)]$ **do**
        $w := Par(w)$;
      end;
      if $node[w].valid$ **then**
        $val := out[w]$; $(*$ Reconstructed node found $*)$
      else if $Par(w) \neq \perp$ **then**
          $val := (Par(w) \neq \perp)\ ?\ out[w] \vee f[Par(w)]\ :\ 0$;
        end;
      end;
      for all u in root path from v to w inclusive **do**
        if $u \in Vp$ **then**
          $out[u] := val$; $node[u].valid :=$ **true**;
        end;
      end;
    end;
  end;
end.

Timings Xeon



Speed-up Xeon



An Improvement

- Compute all markers using parallel erosions and store each scale i in an image $out[i]$;
- Compute a single Max-Tree in parallel, maintaining the maximum marker for all scales i for each node n in $out[i][n]$;
- **Compute reconstruction (in parallel) for scale all scales i, storing the result in $out[i]$;**
- Compute differences between scales (in parallel);
- Output respective instance of DMP vector field.

procedure *MaxTreeReconstruct* (*Vp* : *Section*; **var** $node$: *Max-Tree*; **var** out : *DMP*)
  for all $v \in Vp$ **do**
    if not $node[v].valid$ **then**
      $w := v$;
      for all scales i **increasing order do**
        while $Par(w) \neq \perp \wedge$ **not** $node[w].valid \wedge f[Par(w)] > out[Par(w)]$ **do**
          $w := Par(w)$;
        end;
        $ws[i] := w$; $(*$ temporary storage of for each scale $*)$
        if $node[w].valid$ **then**
          for all scales $j \geq i$ **do**
            $val[j] := out[j][w]; \quad ws[j] := w \quad (*$ Reconstructed node found $*)$
          end;
        else if $Par(w) \neq \perp$ **then**
            $val[i] := out[i][w] \vee f[Par(w)]; \quad (*\ w$ is reconstructed $*)$
          else $(*$ marker was empty $*)$
            for all scales $j \geq i$ **do**
              $val[j] := 0; \quad ws[j] := w;$ **end;**
          end;
        end;
      end;

Reconstruction II(cont.)

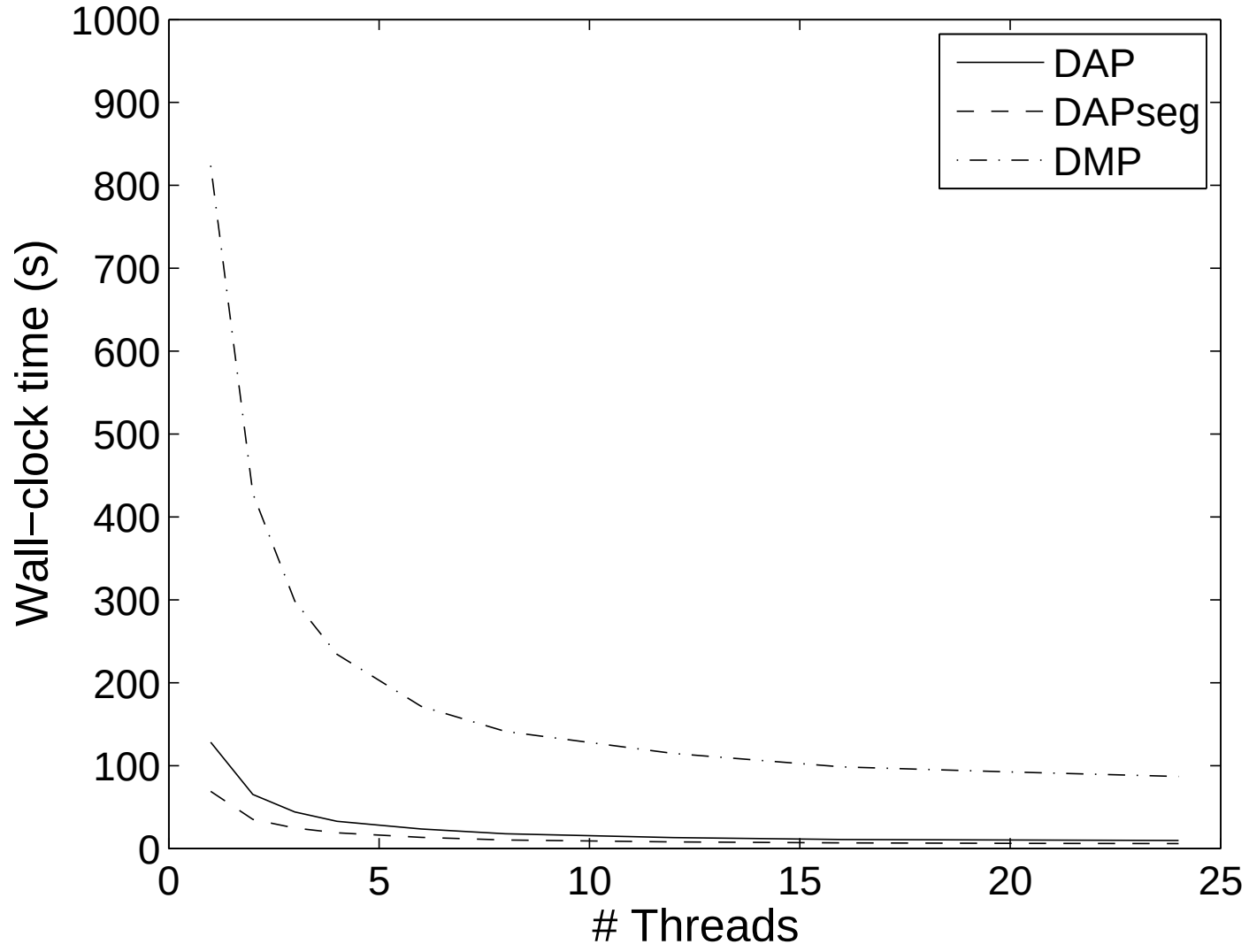

```
        u := v;
        for all scales i increasing order do
            repeat
                if u ∈ Vp then
                    for all scales j < i do out[j][u] := f[u];end;
                    for all scales j ≥ i do  out[j][u] := val[j]; end;
                    node[u].valid := true;
                end;
                u := node[u].parent;
            until  u = ws[i];
        end;
        if u ∈ Vp then   (* Process ws[numscales − 1] *)
            for all scales j < i do out[j][u] := f[u];end;
            for all scales j ≥ i do  out[j][u] := val[j]; end;
            node[u].valid := true;
        end;
    end;
  end;
end.
```

process *parDMP*(p : *ThreadID* ; f : *Image* ; **var** $node$: *Max-Tree* ; **var** out : *DMP*)
$find$ xm $such\ that$ $f(\texttt{xm}) \leq f(x)\ \forall x \in V^p$;
hm := $f(\texttt{xm})$; set[hm] := {xm} ; lero[hm] := xm ;
for all scales s **do** maximum[s] := 0;**end**;
flood(Vp, hm, maximum, f, $node$, out);
var $i := 1$, $q := p$;
while $p + i < K \ \wedge\ q$ **mod** $2 = 0$ **do**
$P(\texttt{sa}[p+i])$ (∗ wait to glue with right-hand neighbor ∗) ;
for all edges (x, y) between $V(p)$ and $V(p+i)$ **do**
connect$(x, y, f, node)$;
end ;
$i := 2 * i$; $q := q/2$;
end ;
if $p = 0$ **then** (∗ release the waiting threads ∗)
for $i := 1$ **to** $K - 1$ **do** $V(\texttt{sb}[i])$ **end**
else
$V(\texttt{sa}[p])$; $P(\texttt{sb}[p])$; (∗ signal left-hand neighbor; wait for thread 0 ∗)
end ;
MaxTreeReconstruct(*Vp*, $node$, out) ;
end.

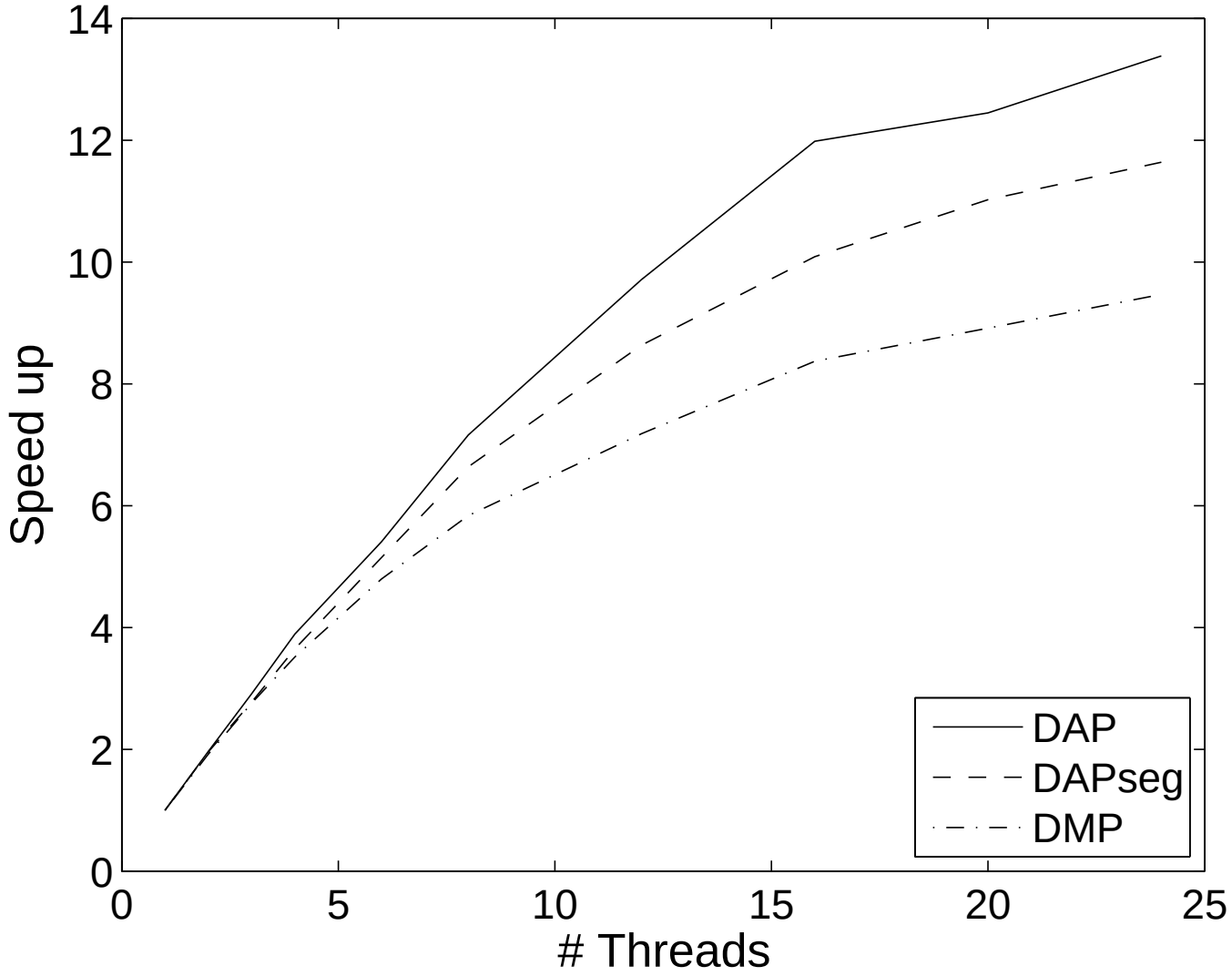
Different Profiles

- Images: 120 MB and 1.2 GB
- Parallel Algorithms:
  - Differential Morphological Profile (DMP)
  - Differential Area Profile (DAP)
  - DAP-segmentation
- The Machine
  - Part of Millipede cluster
  - quad socket, opteron machine, 6 cores per socket
  - 128 GB memory

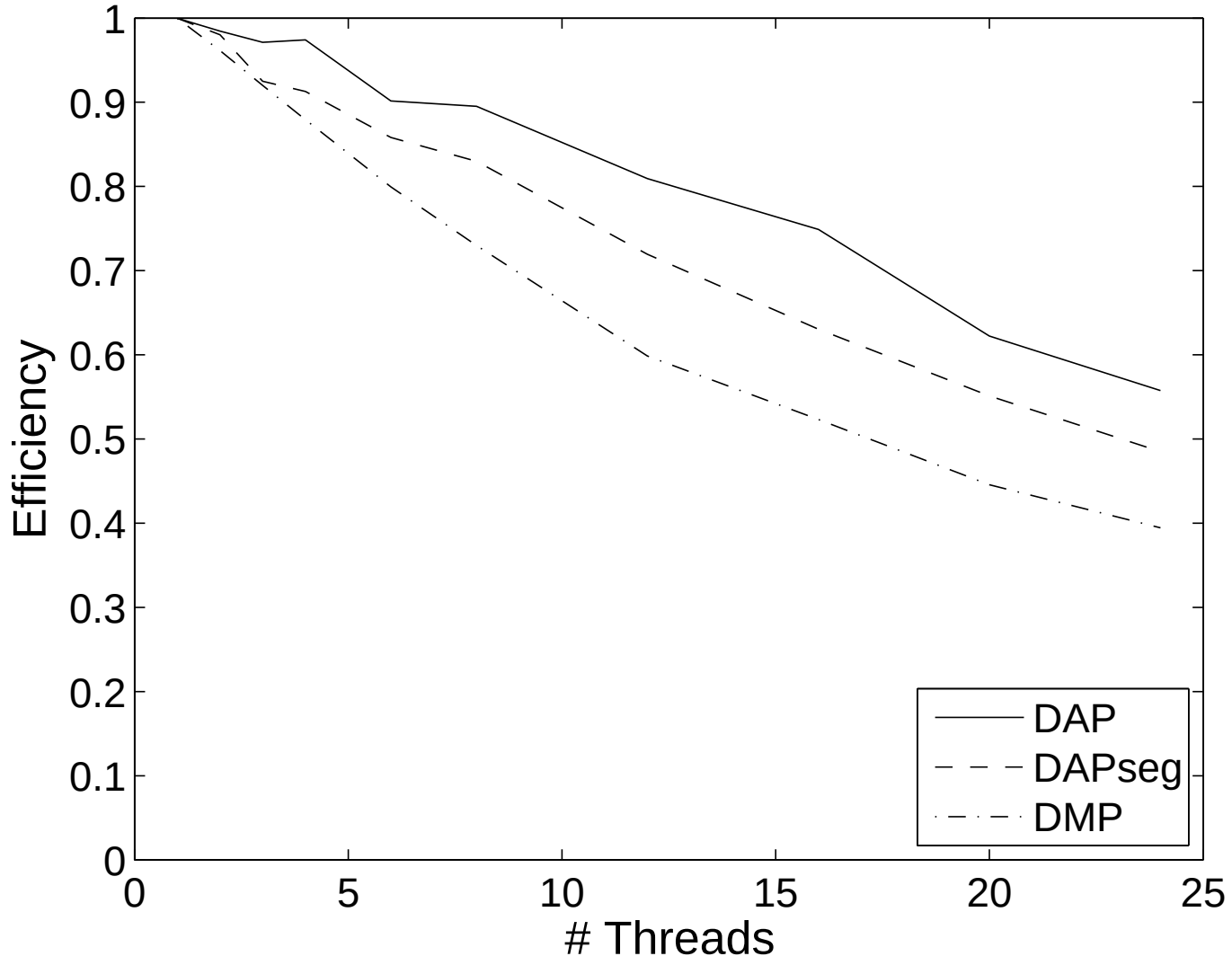
Timings


1000
900
800
700
600
500
400
300
200
100
0
Wall−clock time (s)
0
5
10
15
20
25
Threads
DAP
DAPseg
DMP

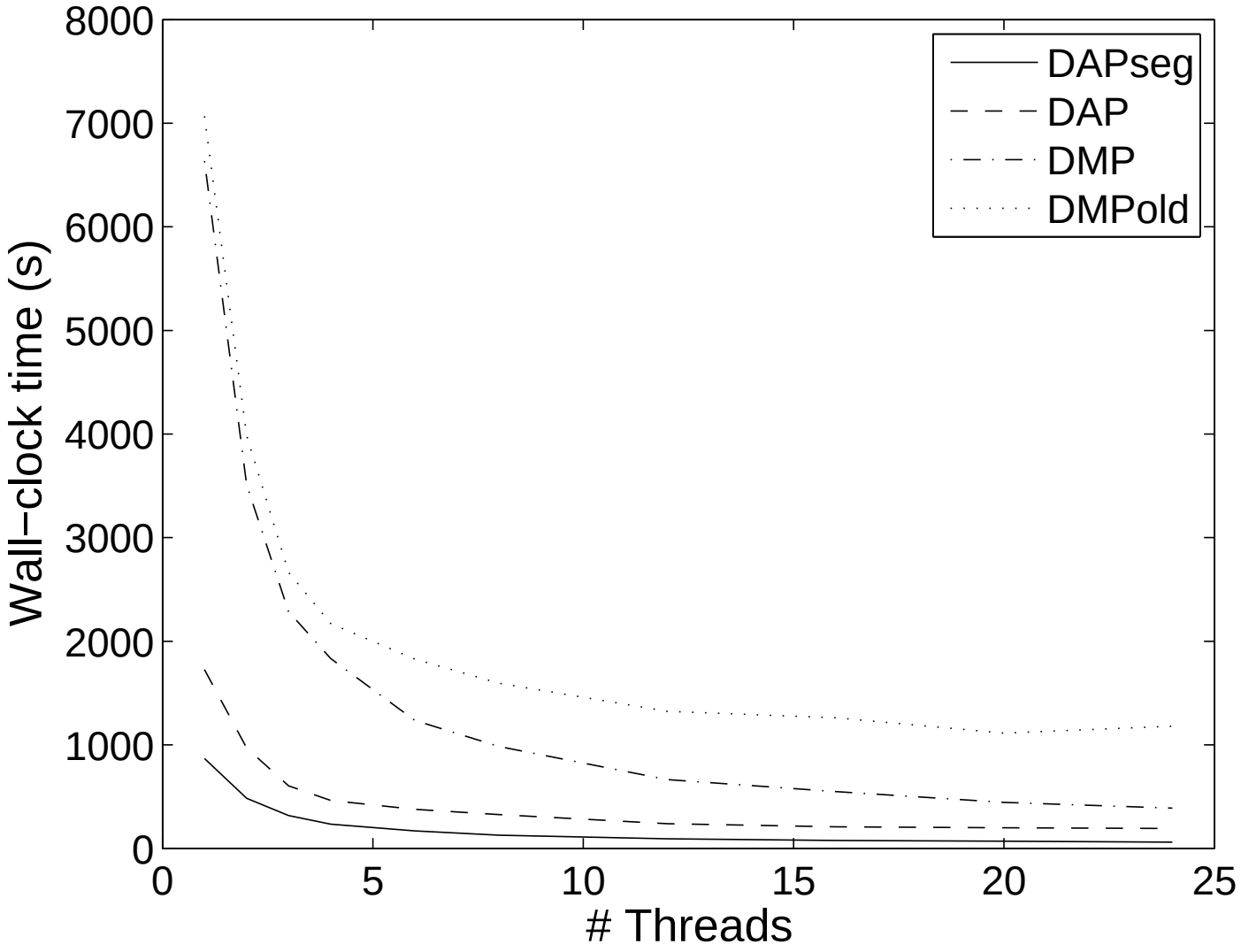
Speed-up



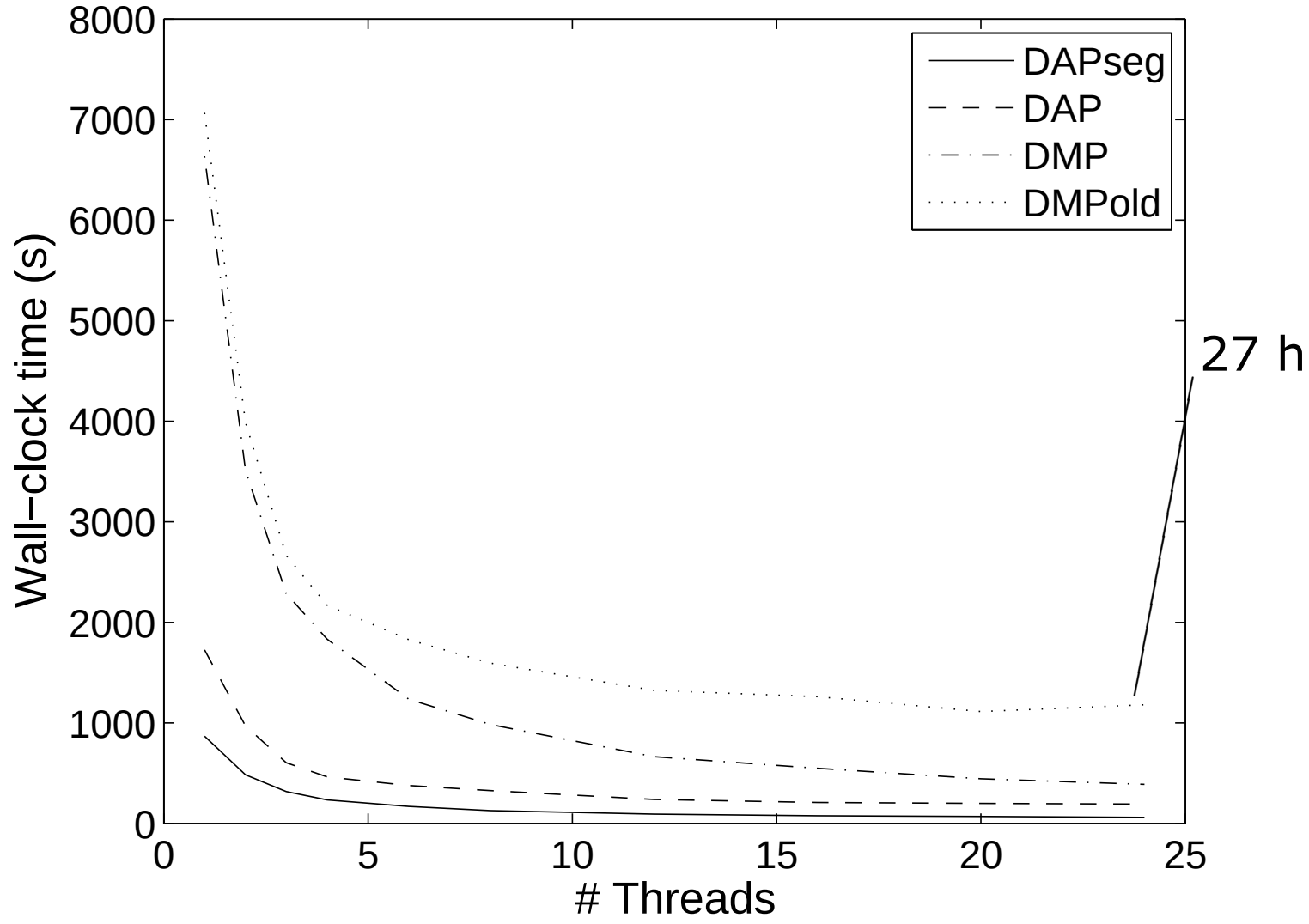
Efficiency


Efficiency
Threads
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
DAP
DAPseg
DMP

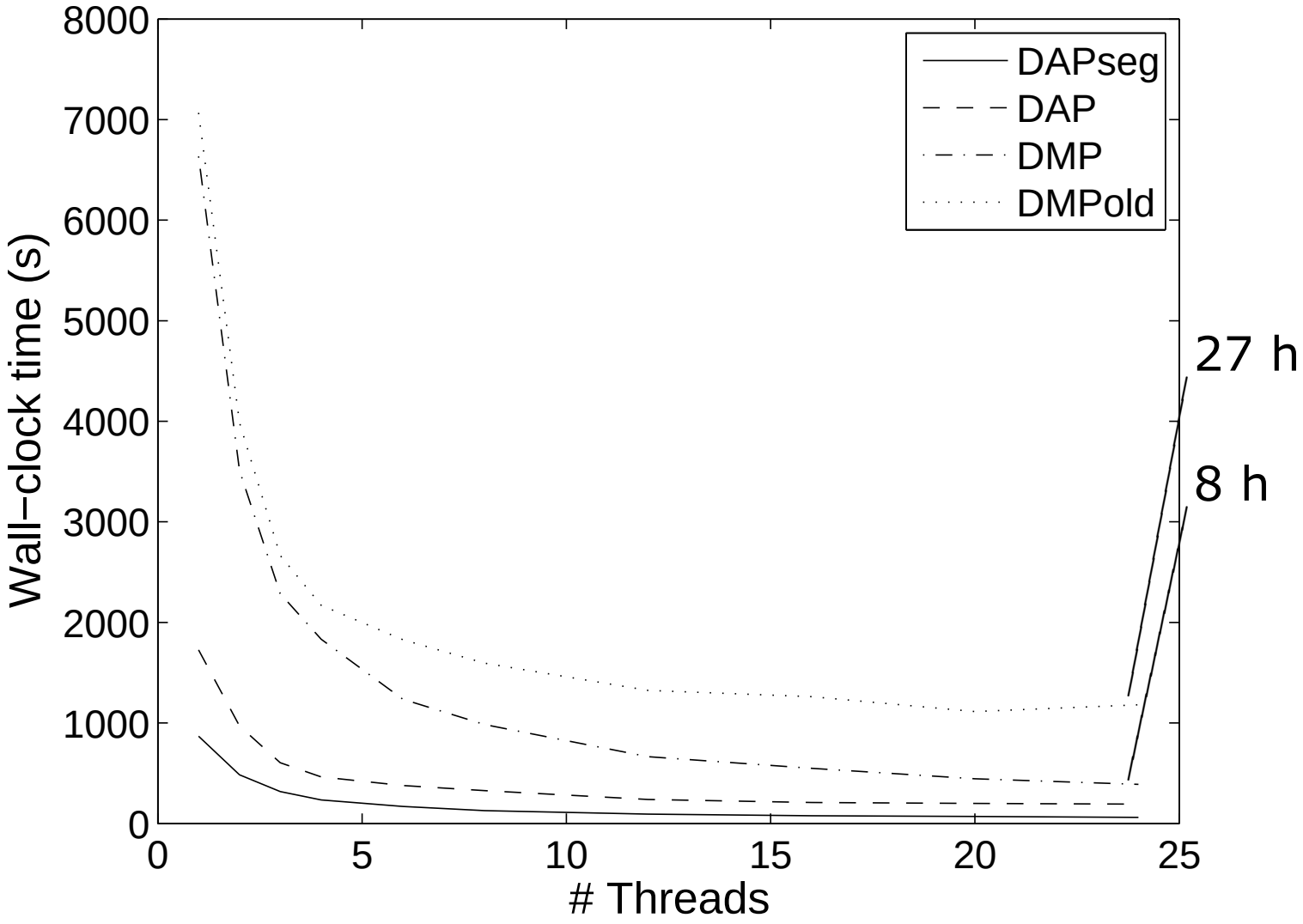
Timings 1.2GB



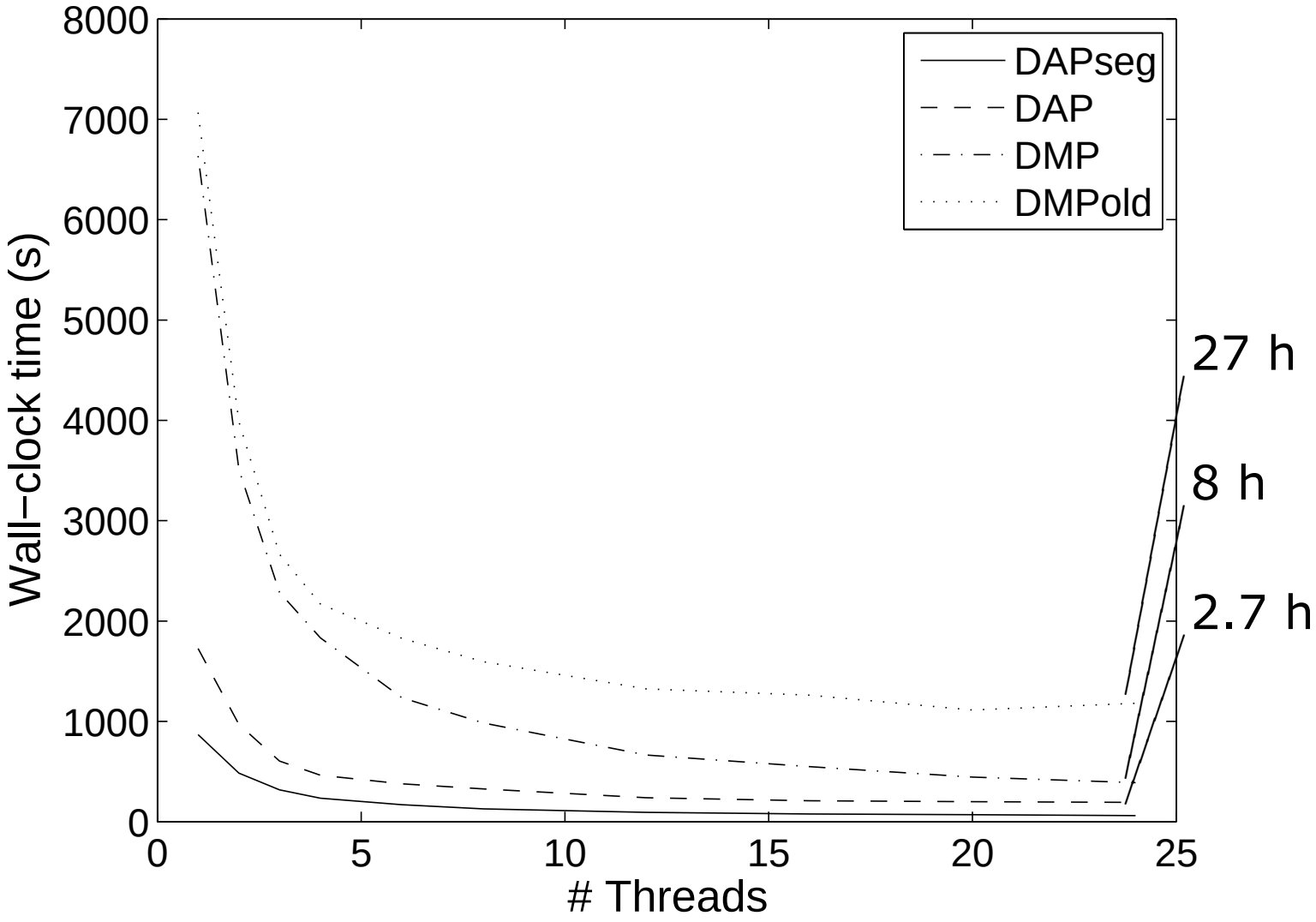
Timings 1.2GB



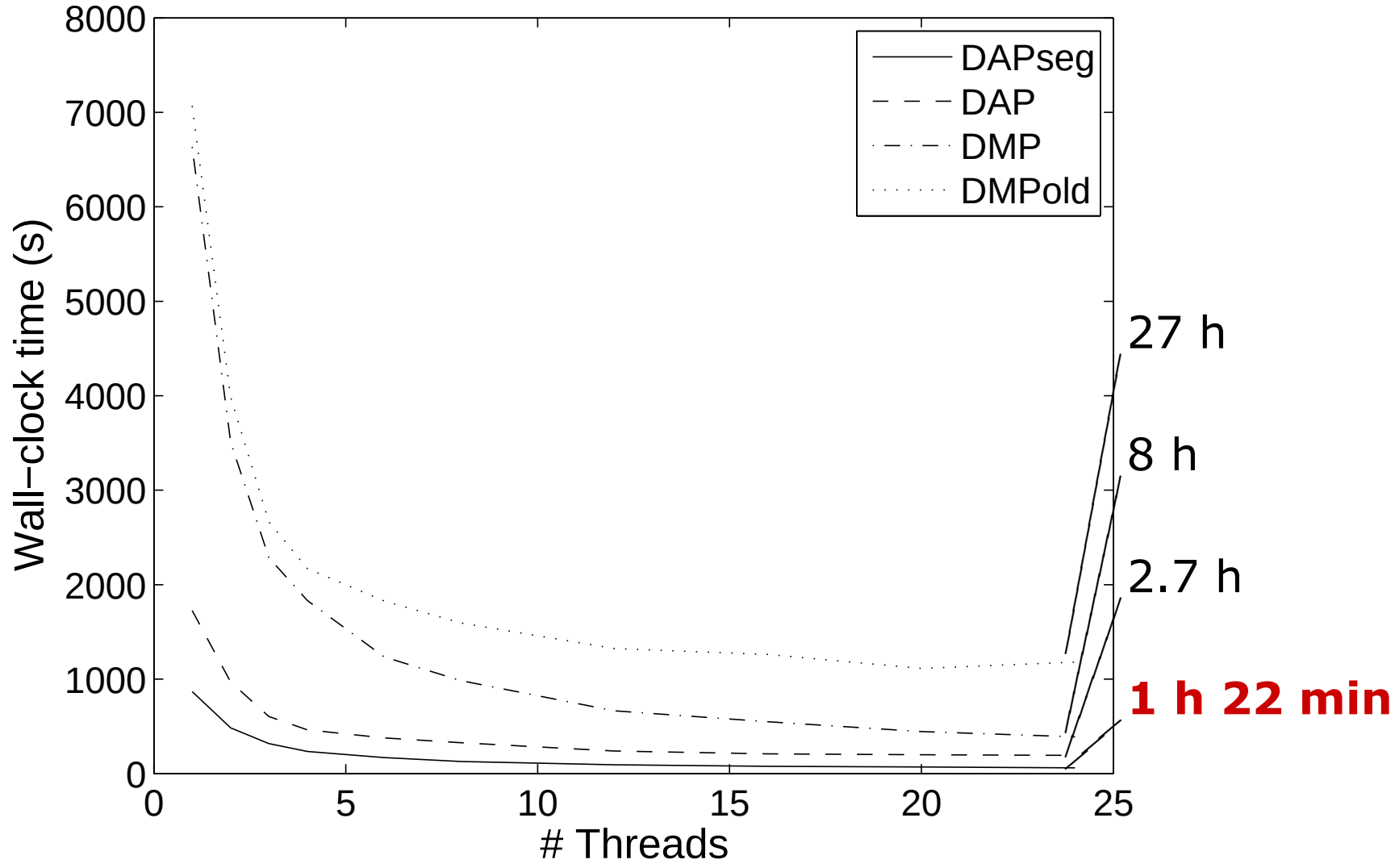
Timings 1.2GB



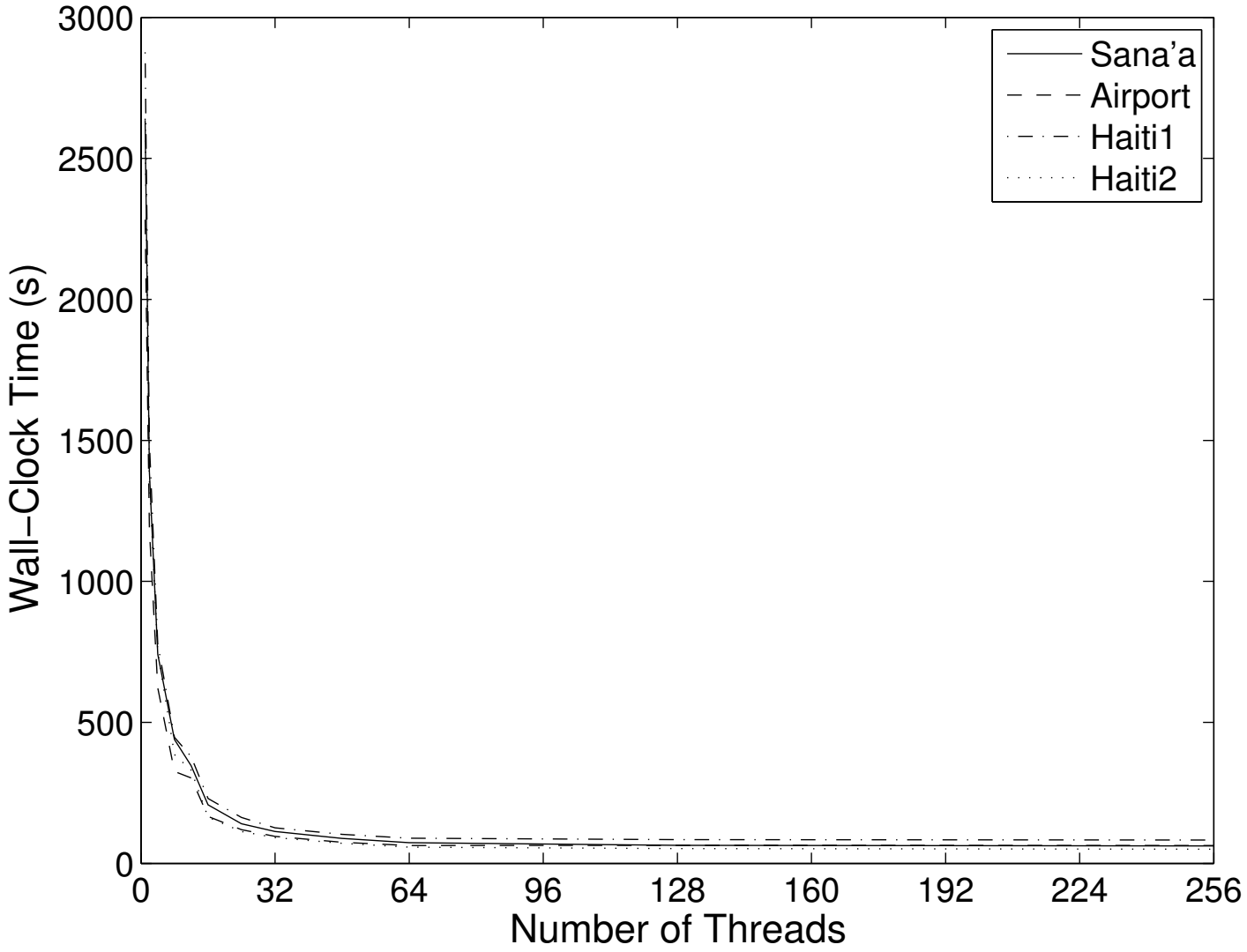
Timings 1.2GB



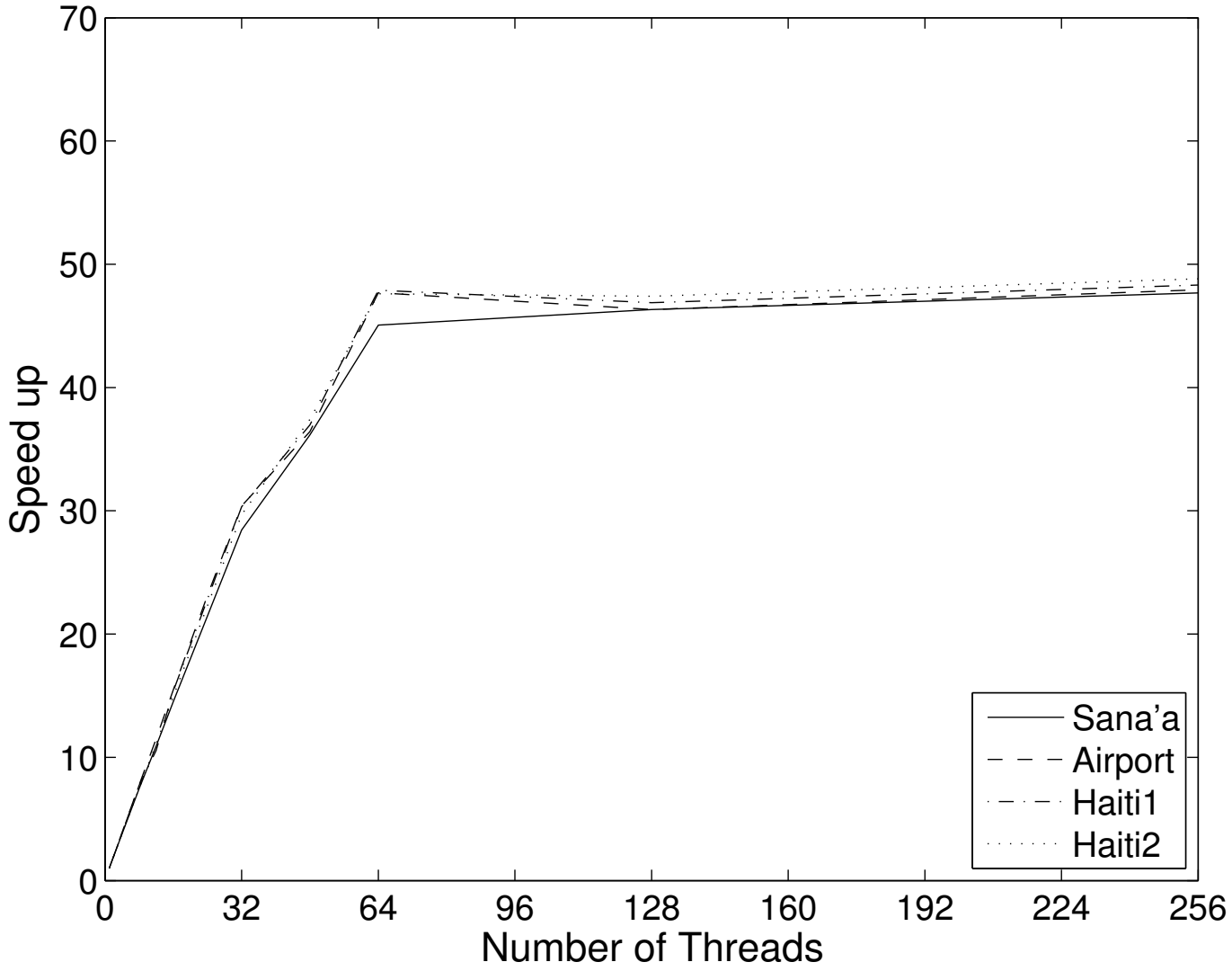
Timings 1.2GB



Bigger machines: 64 cores



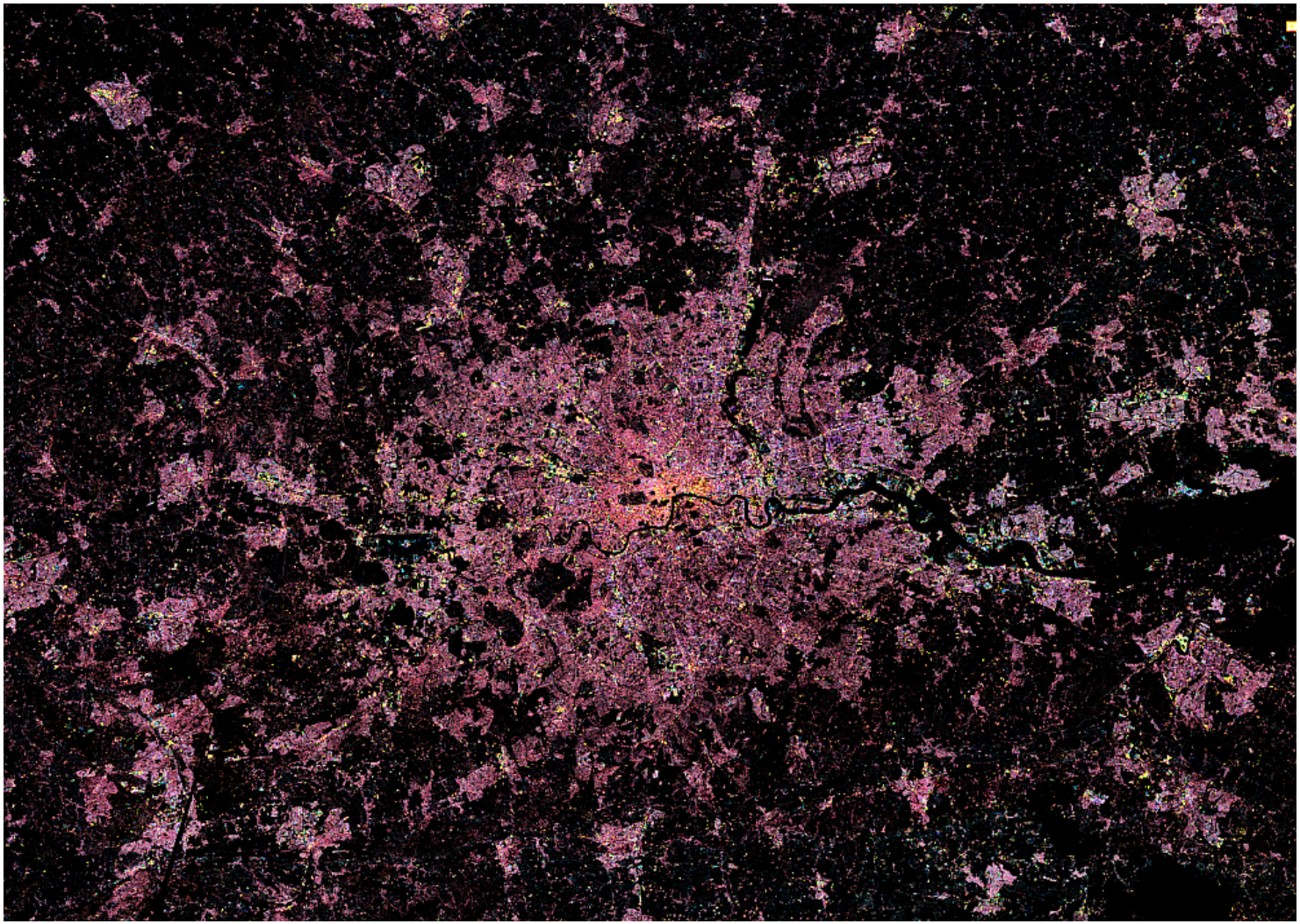
Bigger machines: 64 cores, part II



The Global Human Settlement Layer

London, UK. © Landsat; 2012.

The Global Human Settlement Layer



London, UK. © European Commission; Joint Research Centre; 2012.

The Global Human Settlement Layer

London, UK. Night-lights by Andre Kuiper. Credit ESA/NASA. Posted at Kuipers' photostream in Flickr.com. Taken from the ISS on April 5, 2012 using a Nikon D3S camera.

But wait, there's more!

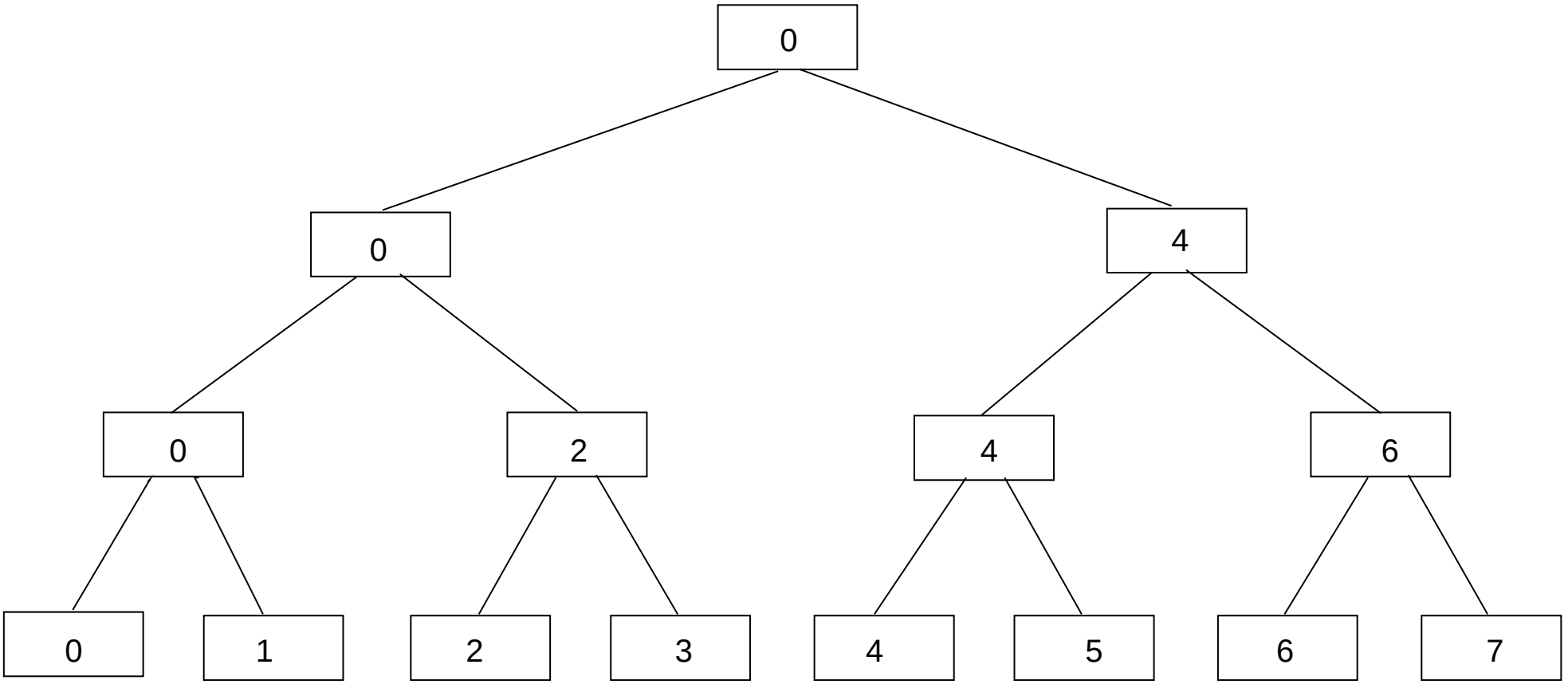
More data of course!

- Modern stitching methods generate image $\gg$ 10 Gpixel
- Shared memory sizes insufficient
- Astronomical data sets astronomical in size
- Haiti aerial image data set 1.5 Tpixel

How can we handle these?

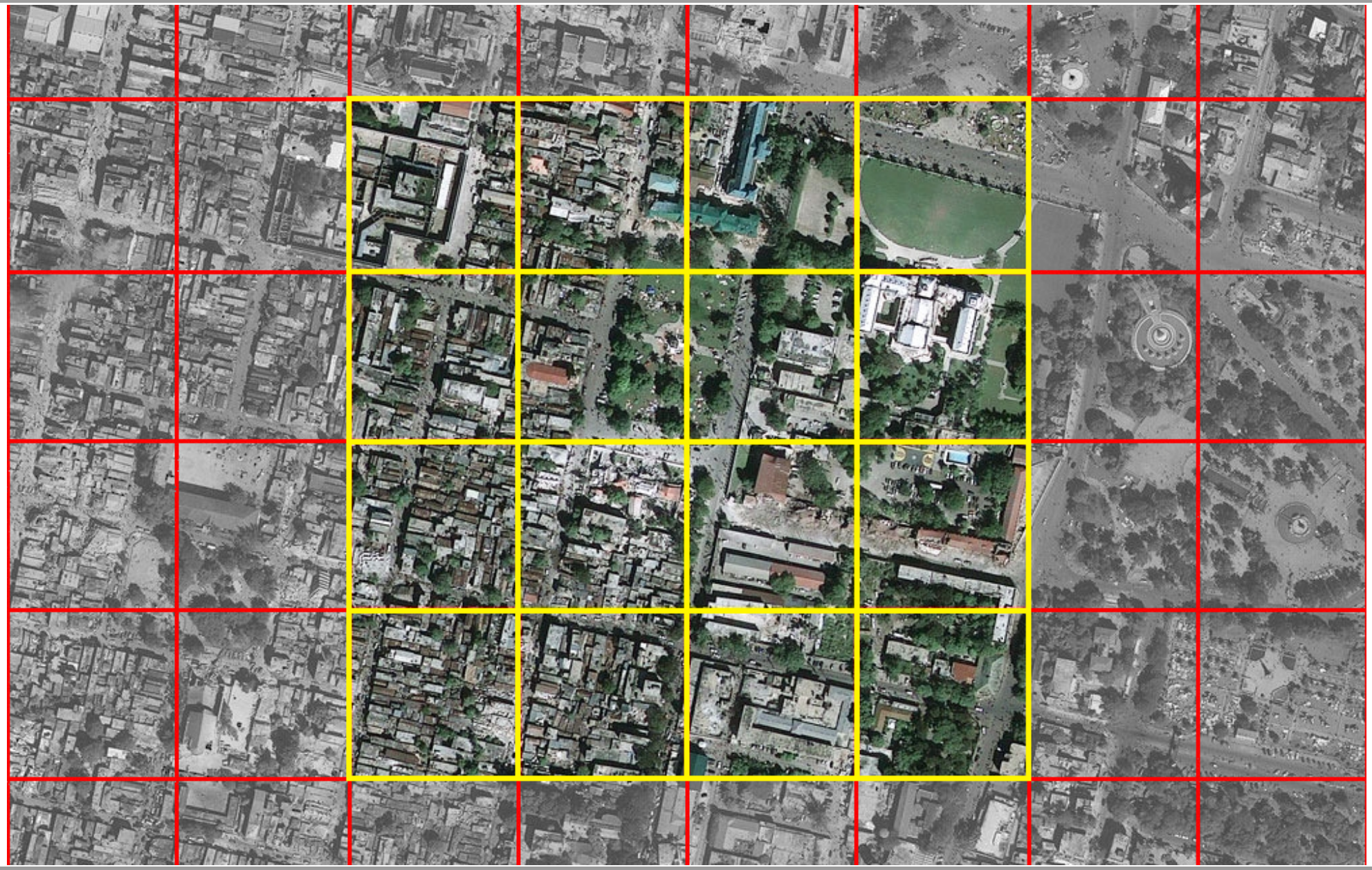
Folding@Home?

MapReduce?



Algorithm ≠ MapReduce

Algorithm ≠ MapReduce



Boundary trees

A boundary tree of an image block $B_{i,j}$ is

- a subset of the Max-Tree of $B_{i,j}$
- containing only nodes C_h^k for which P_h^k touches the boundary $\delta B_{i,j}$ of $B_{i,j}$

Observations:

- During merging of max-trees of neighbouring blocks *only the boundary tree nodes are affected*
- Therefore, only boundary trees need be merged

Distributed Memory Algorithm

- Compute Max-Trees of all blocks $B_{i,j}$
- Extract boundary trees
- Hierarchically merge boundary trees up to root
- Hierarchically transmit updated boundary trees down to leaves
- Update Max-Tree of each block $B_{i,j}$

Boundary-Tree Size

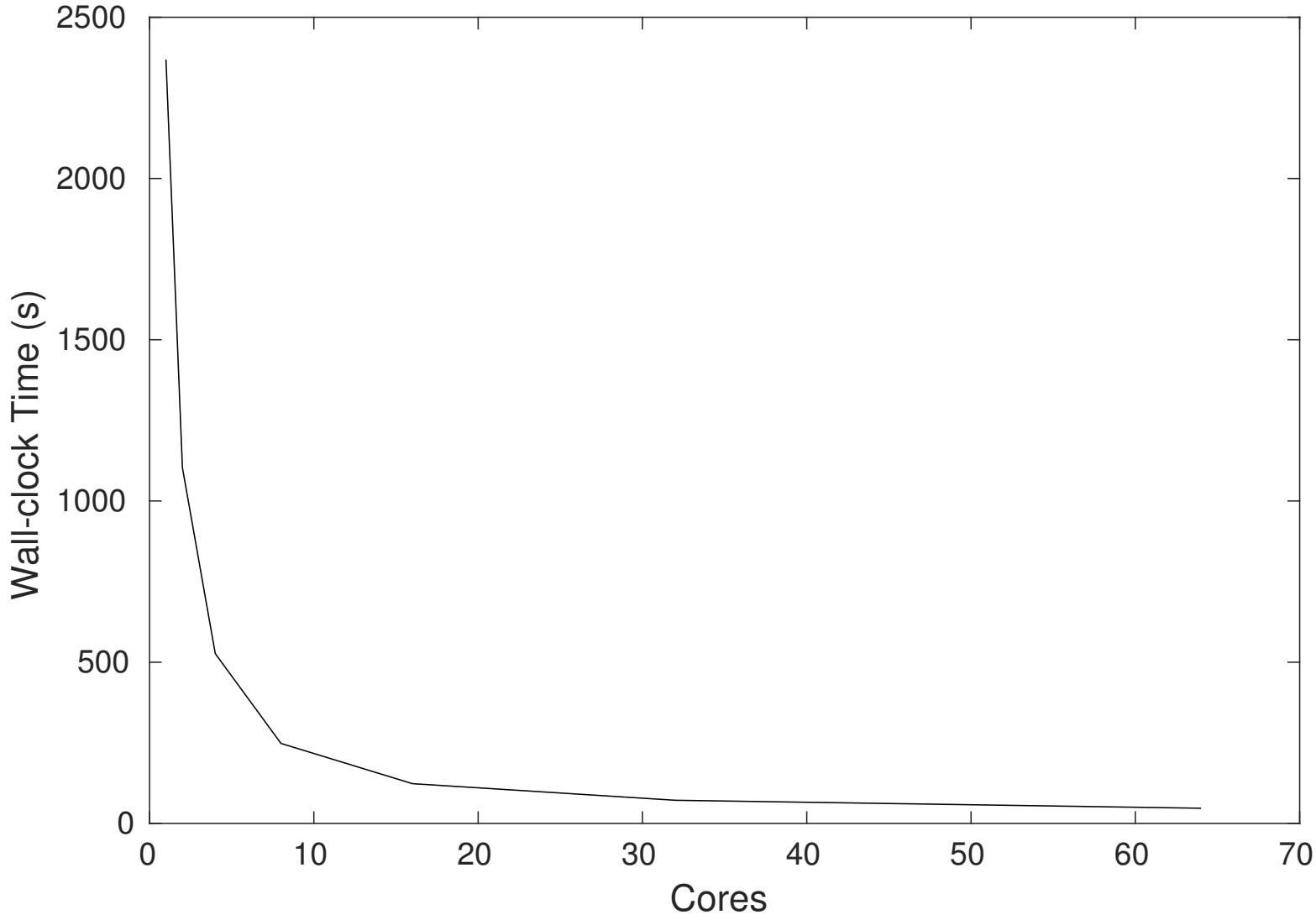
- The boundary $\delta B_{i,j}$ contains at most $\frac{|\delta B_{i,j}|}{2}$ local maxima
- The number of nodes is at most $\min(\frac{|\delta B_{i,j}|}{2}G, |B_{i,j}|)$
- Provided $\frac{|\delta B_{i,j}|}{2}G < |B_{i,j}|$ we have *reduction*
- For $G = 256$ and $|B_{i,j}| = 40960 \times 40960 = 1.6$ Gpixel we have $\frac{|\delta B_{i,j}|}{2}G = 512 \times 40960$ or $80\times$ reduction

The Impact (8-bits)

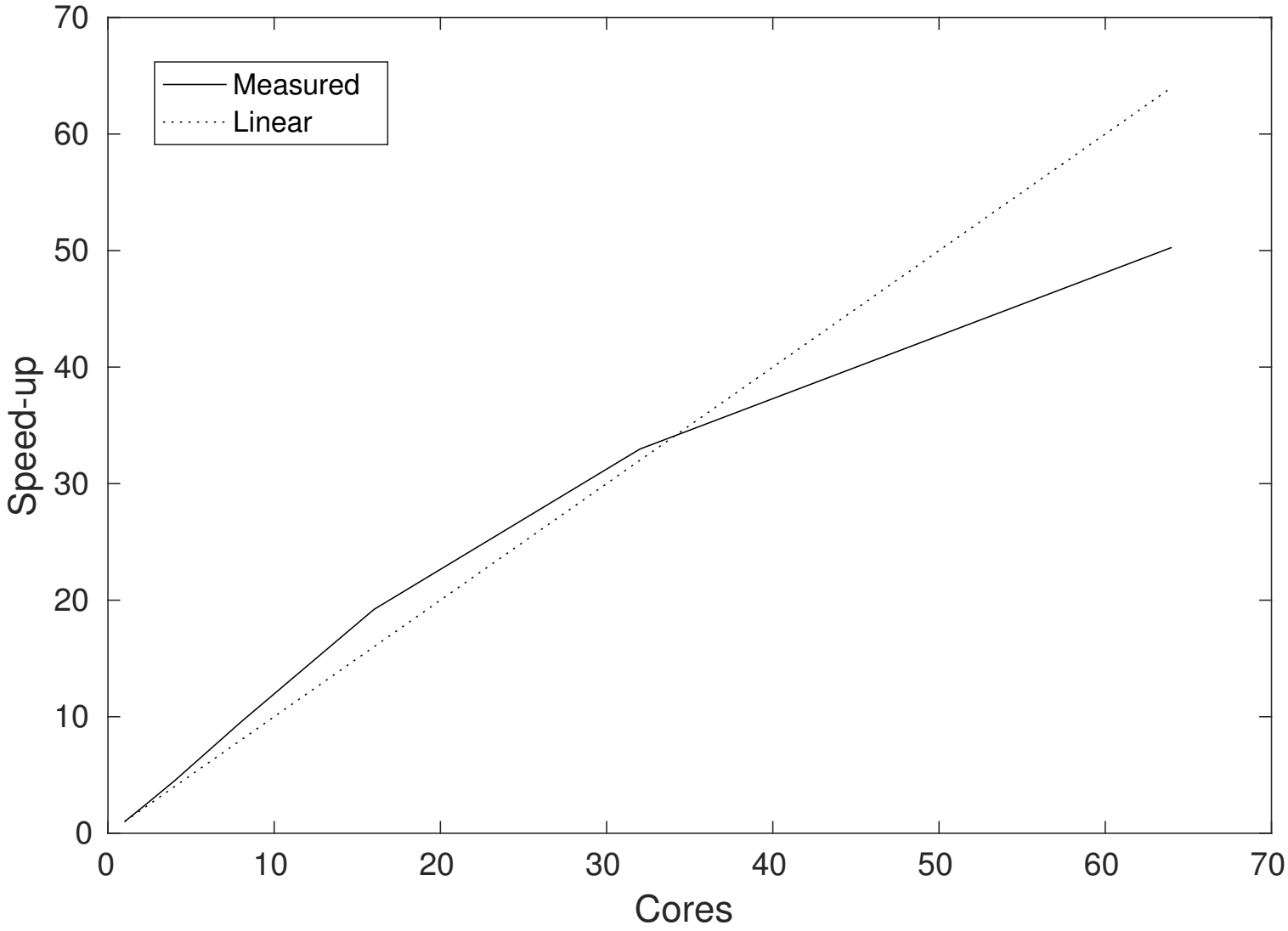
iteration	Imsize	$\lvert B_{i,j} \rvert$	$\frac{\lvert \delta B_{i,j} \rvert}{2} G$	Mem (orig)	Mem (new)
1	6.4 G	1.6 G	20 M	128.0 GB	33.6 GB
2	25.6 G	6.4 G	40 M	512.0 GB	35.2 GB
3	102.4 G	25.6 G	80 M	2048.0 GB	38.4 GB
4	409.6 G	102.4 G	160 M	8192.0 GB	44.8 GB
5	1.6 T	409.6 G	320 M	32.0 T	57.6 GB
6	6.4 T	1.6 T	640 M	128.0 T	83.2 GB
7	25.6 T	6.4 T	1280 M	512.0 T	134.4 GB

- We implemented the algorithm in MPI
- First test were done on a 3.8 Gpixel image on a 64-core compute server
- Three experiments were done on the Peregrine cluster
  - Hierarchically dividing a 1.2 Gpixel image into 1, 2, . . .,128 tiles, using one core per node
  - Using 1, 2, . . ., 128 tiles of 1.2 Gpixel to create increasing image sizes, again one core per node
  - Same as above, but splitting each 1.2 Gpixel tile into 16, using 1, 2, . . ., 32 nodes, using 16 cores per node

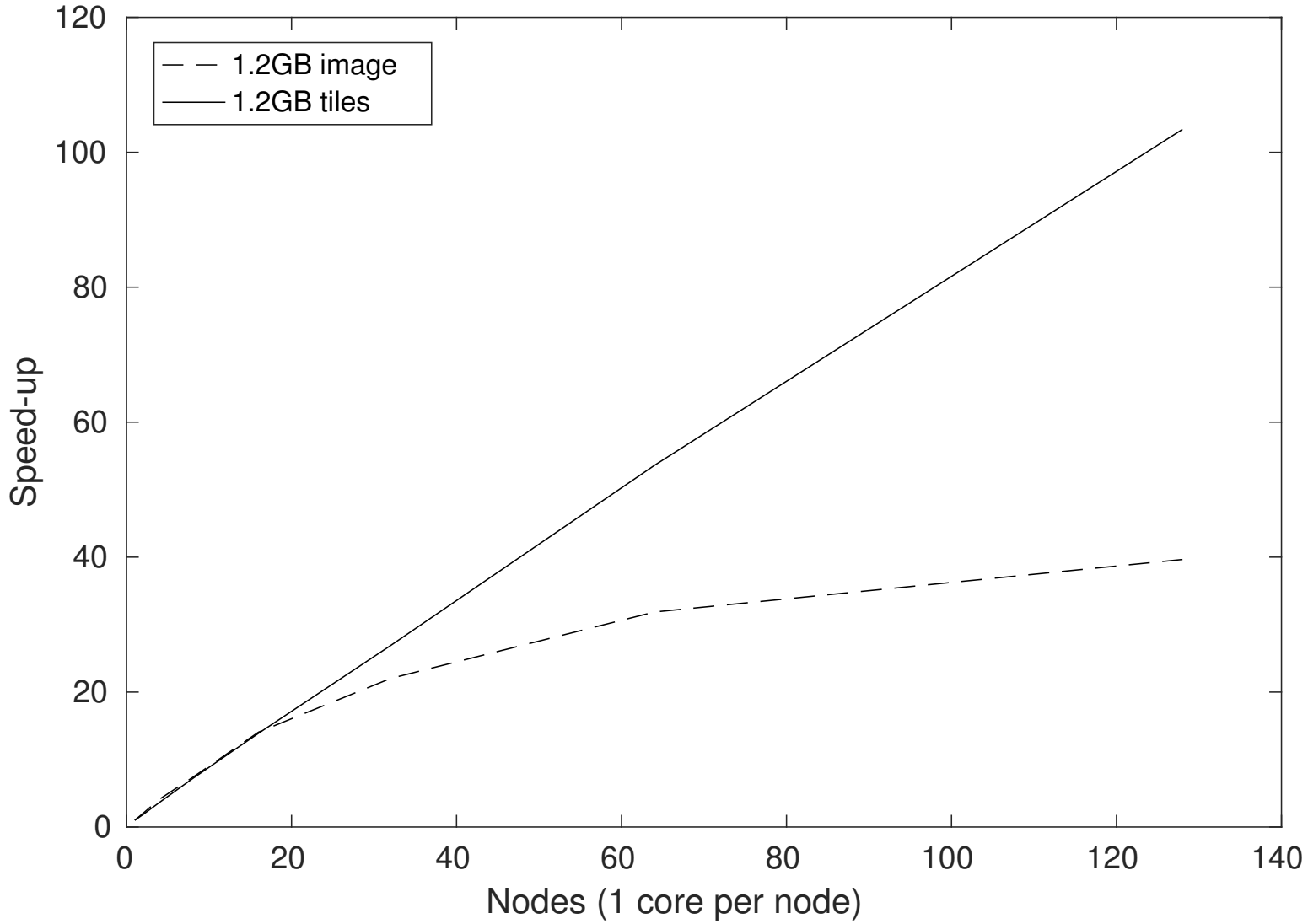
Results: Timings on Zeus



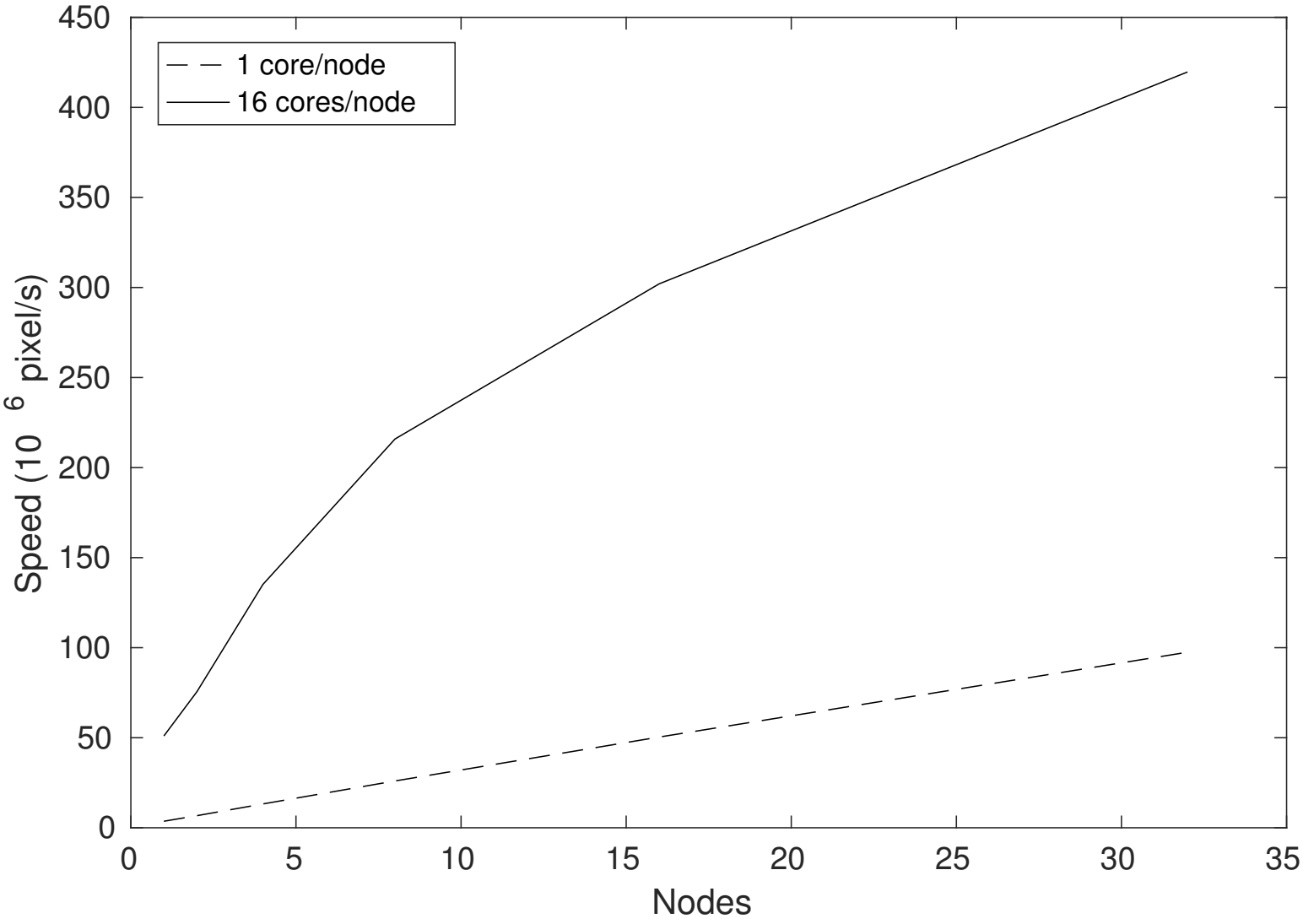
Results: Speed-Up on Zeus



Results: Speed-Up on Peregrine



Results: Speed on Peregrine



The Impact (16 bit)

iteration	Imsize	$\lvert B_{i,j}\rvert$	$\frac{\lvert\delta B_{i,j}\rvert}{2}G$	Mem (orig)	Mem (new)
1	6.4 G	1.6 G	5.12 G	128.0 GB	37.12 GB
2	25.6 G	6.4 G	10.24 G	512.0 GB	42.24 GB
3	102.4 G	25.6 G	20.48 G	2048.0 GB	52.48 GB
4	409.6 G	102.4 G	40.96 G	8192.0 GB	72.96 GB
5	1.6 T	409.6 G	81.92 G	32.0 T	103.92 GB
6	6.4 T	1.6 T	163.84 G	128.0 T	195.84 GB
7	25.6 T	6.4 T	327.68 G	512.0 T	359.68 GB

Looks like 16 bits per pixel is possible!

Where are we now

- An 8-bits per pixel MPI implementation has been built
- It works on a shared memory machine
- Parity in performance and speed-up with shared memory algorithm (Wilkinson et al 2008)
- Tests on cluster show huge performance up to 128 nodes
- Same approach should work on α-partition tree
- 16 bits per pixel should be doable
- Out-of-memory computation possible
- What remains is high-dynamic-range distributed-memory algorithm

?