Benchmarking the CRBLASTER Computational Framework on a 350-MHz 49-core Maestro Development Board

Kenneth Mighell

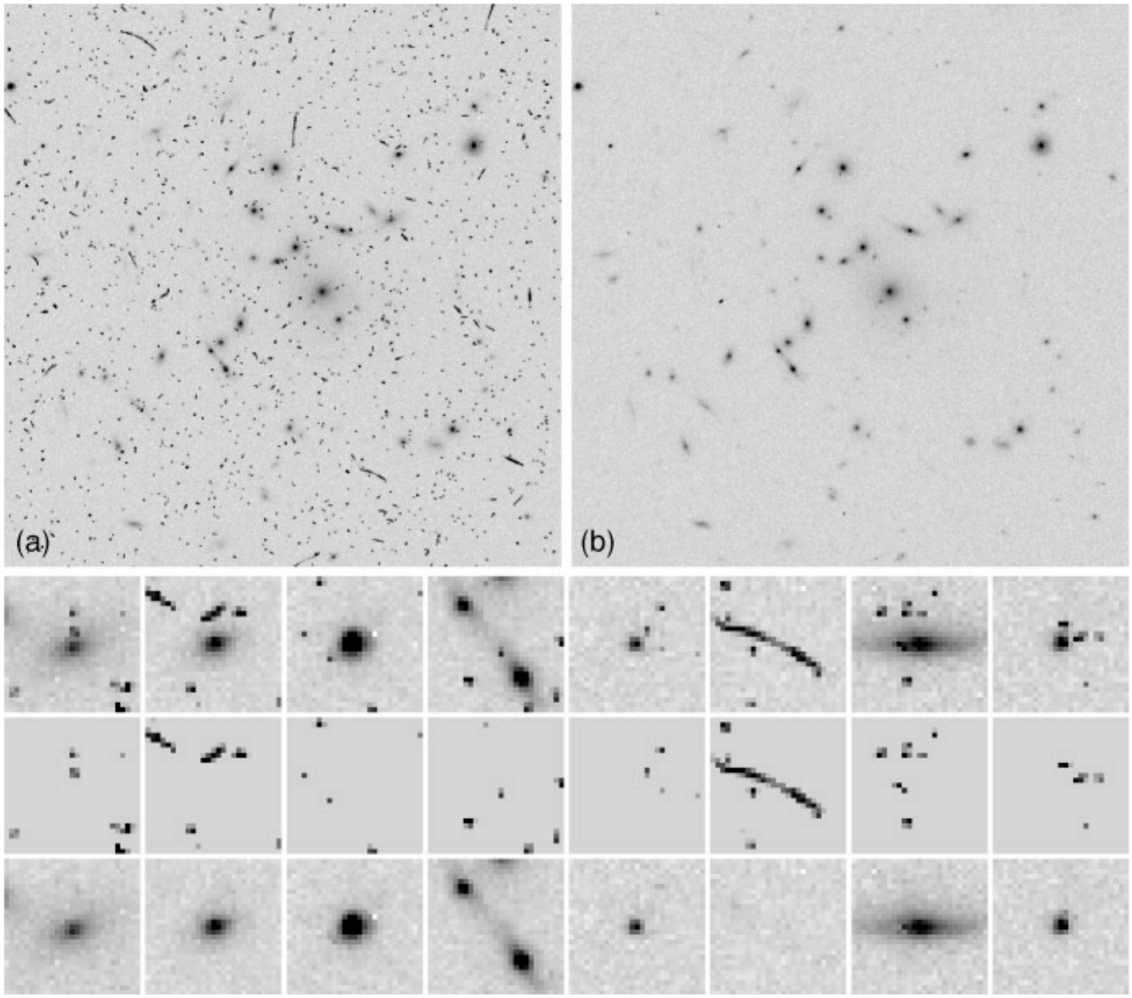
National Optical Astronomy Observatory

ADASS XXI

Paris Marriott Rive Gauche Conference Center, Paris

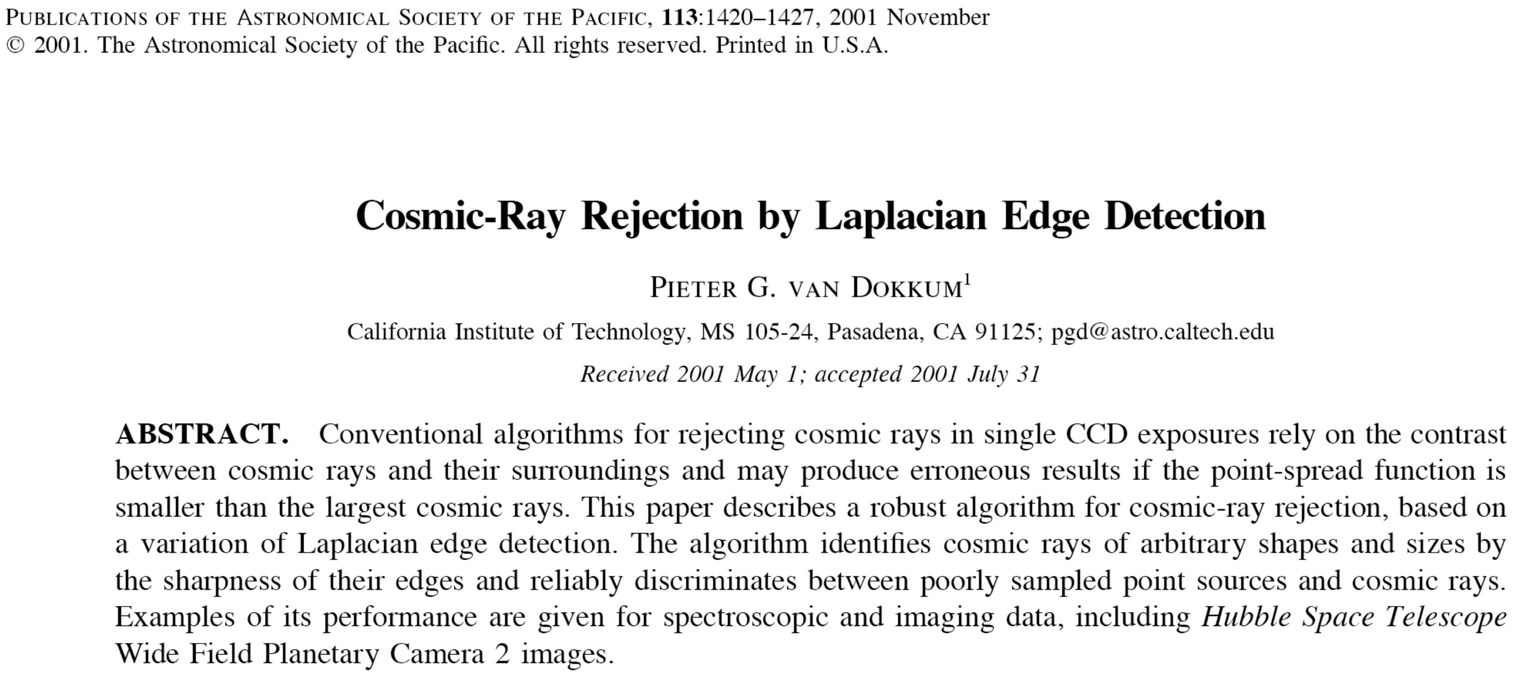
November 8, 2011

The damaging effects of cosmic rays on CCD observations



van Dokkum (2001)

(*a*) *HST* WFPC2 image of galaxy cluster MS 1137+67. The restoration by L.A.COSMIC is shown in (*b*). The small panels show close-ups for a selection of stars and galaxies in various WFPC2 images. The algorithm leaves stars intact and is able to remove cosmic rays of arbitrary shapes and sizes.



PUBLICATIONS OF THE ASTRONOMICAL SOCIETY OF THE PACIFIC, **113**:1420–1427, 2001 November

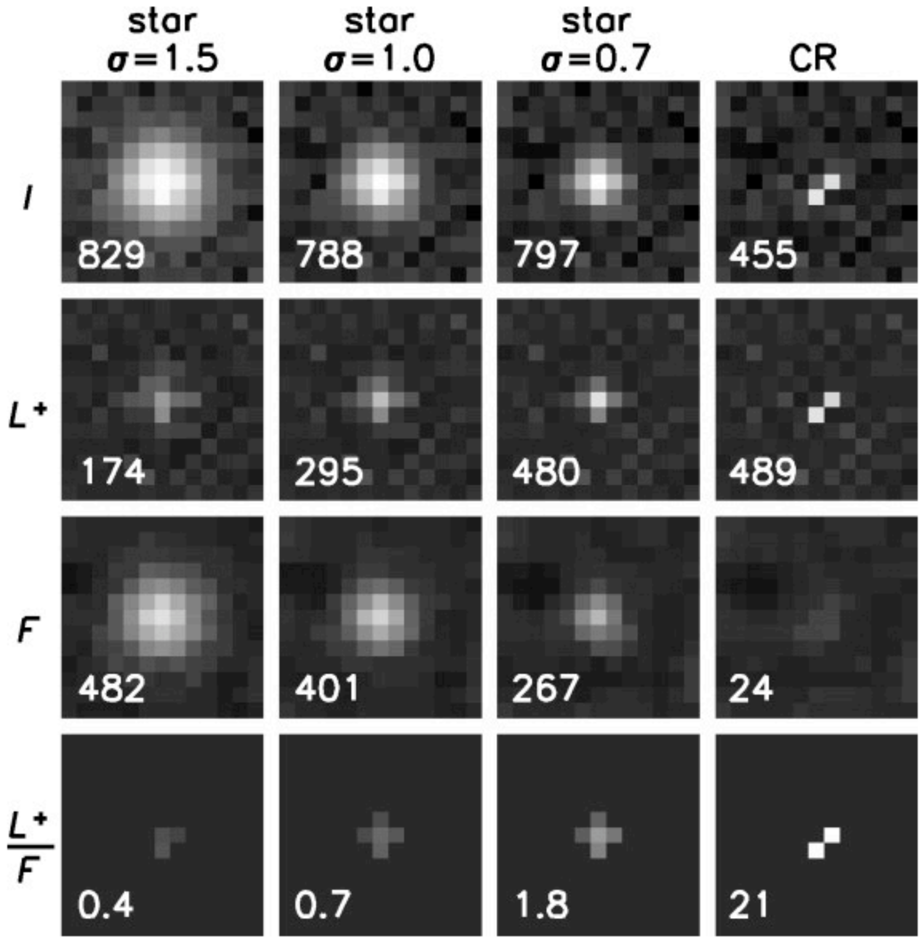
Cosmic-Ray Rejection by Laplacian Edge Detection

PIETER G. VAN DOKKUM[1]

California Institute of Technology, MS 105-24, Pasadena, CA 91125; pgd@astro.caltech.edu

Received 2001 May 1; accepted 2001 July 31

ABSTRACT. Conventional algorithms for rejecting cosmic rays in single CCD exposures rely on the contrast between cosmic rays and their surroundings and may produce erroneous results if the point-spread function is smaller than the largest cosmic rays. This paper describes a robust algorithm for cosmic-ray rejection, based on a variation of Laplacian edge detection. The algorithm identifies cosmic rays of arbitrary shapes and sizes by the sharpness of their edges and reliably discriminates between poorly sampled point sources and cosmic rays. Examples of its performance are given for spectroscopic and imaging data, including *Hubble Space Telescope* Wide Field Planetary Camera 2 images.



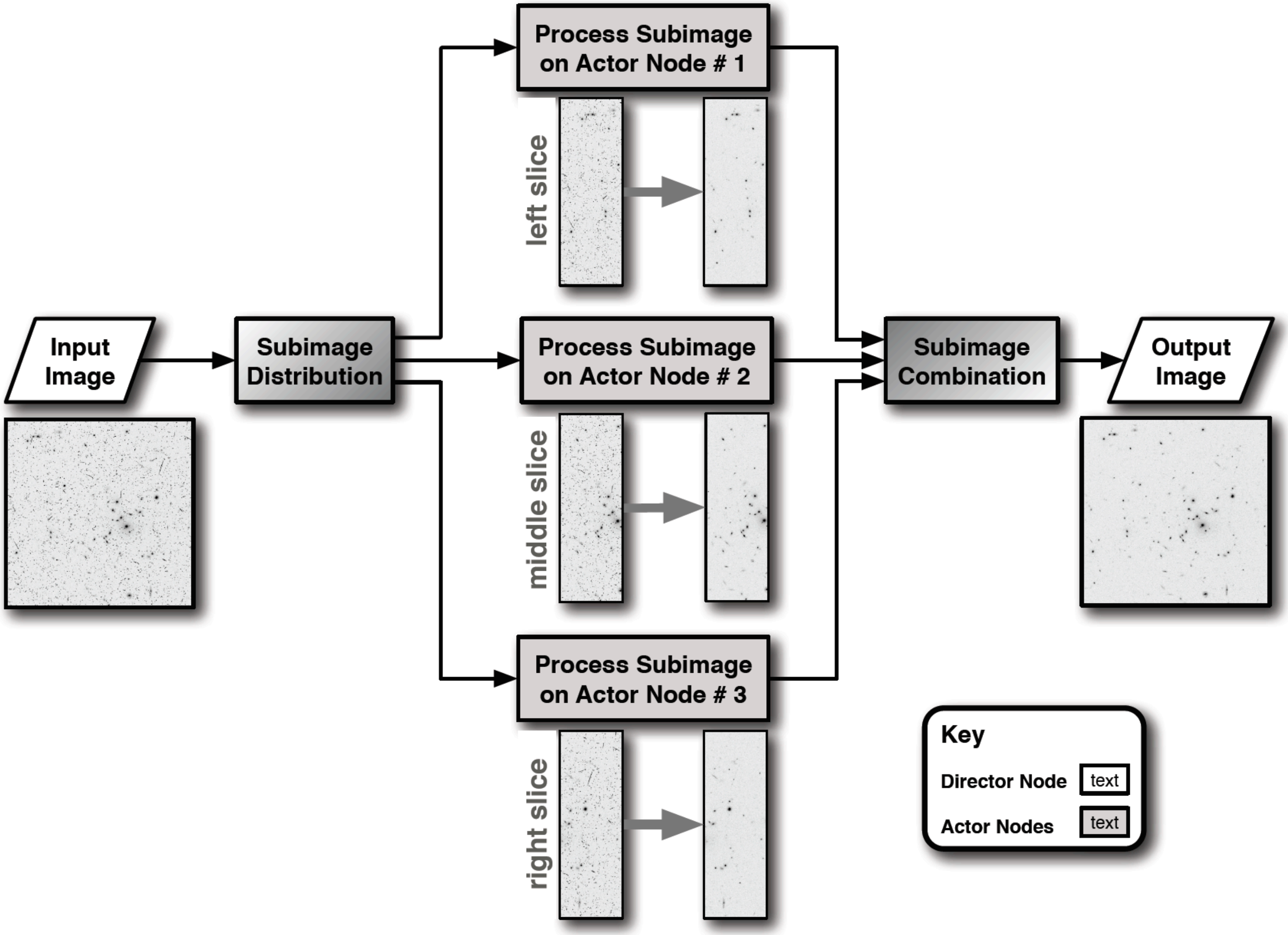
van Dokkum (2001)

Differentiating between marginally sampled point sources and cosmic rays. The panels show, from top to bottom, artificial images of stars and a cosmic ray (I), the Laplacian of these images ($\mathcal{L}^+$), their fine-structure image ($\mathcal{F}$), and the Laplacian divided by the fine structure ($\mathcal{L}^+/\mathcal{F}$). The number in each panel is the value of the highest pixel. The highest pixels in the Laplacian images of the undersampled star ($\sigma = 0.7$ pixels) and the cosmic ray are similar. However, they are very different after division by the fine-structure image.

Problem: Van Dokkum's L.A.COSMIC algorithm is **slow** when using the original IRAF script *lacos_im.cl* .

Solution: A C version running in parallel might be a whole lot faster.

CRBLASTER Parallelization: Embarrassingly Parallel

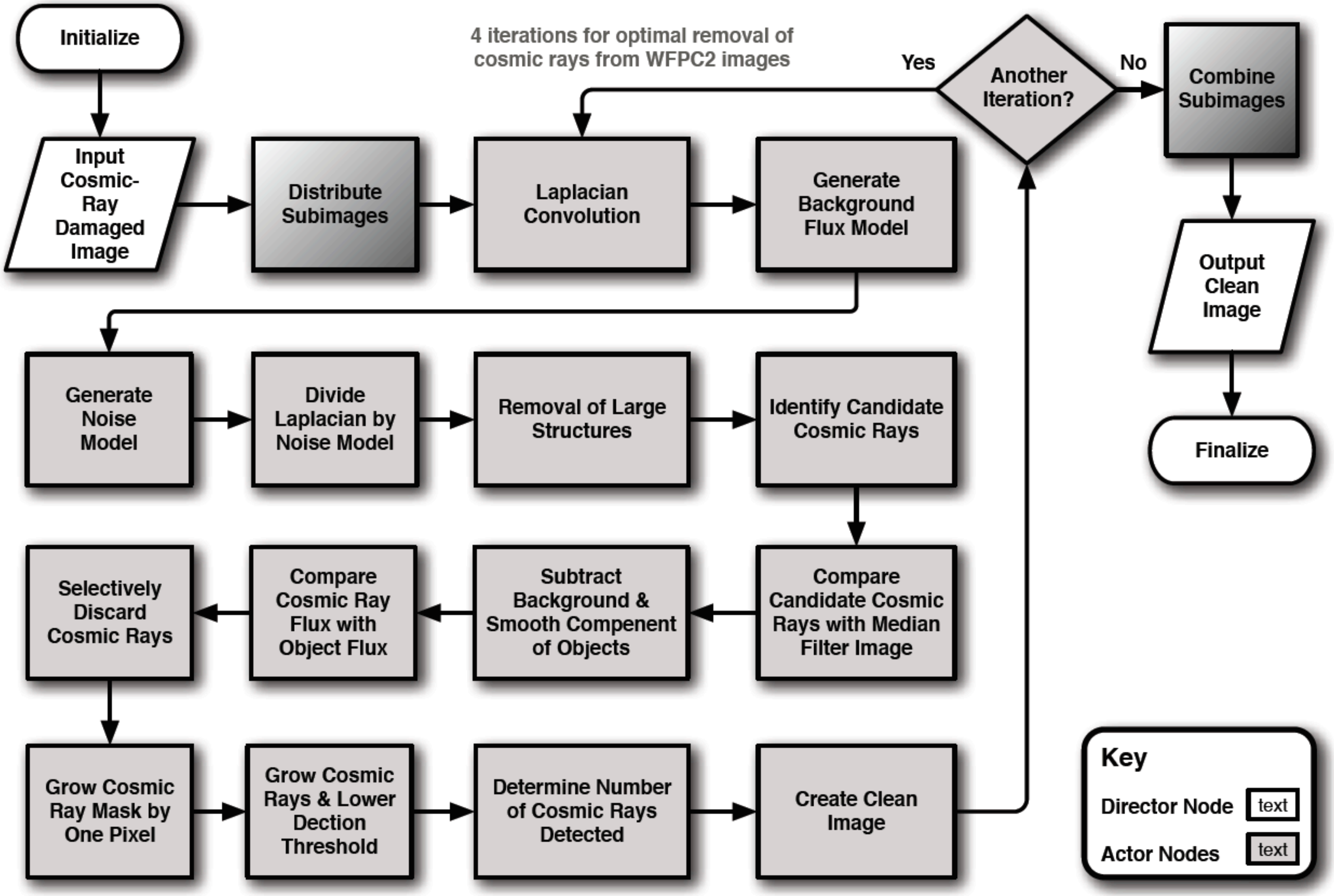

Input Image
Subimage Distribution
Process Subimage on Actor Node # 1
left slice
Process Subimage on Actor Node # 2
middle slice
Process Subimage on Actor Node # 3
right slice
Subimage Combination
Output Image

Key

Director Node	text
Actor Nodes	text

Mighell (HPEC 2010)

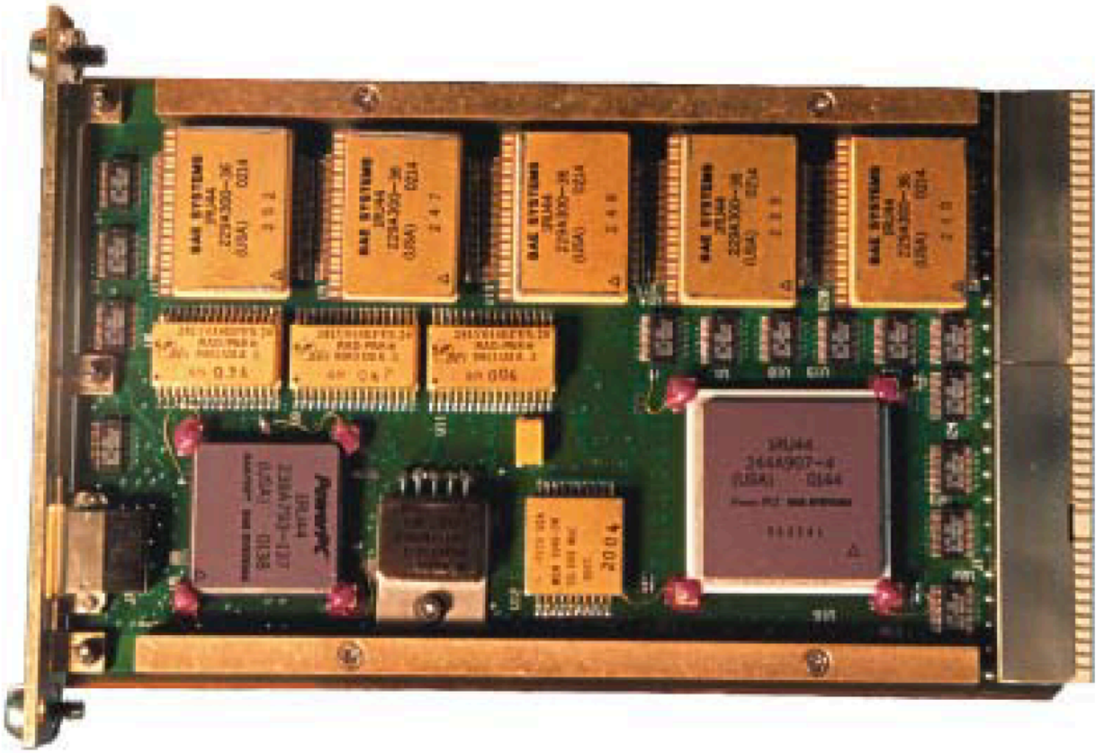
CRBLASTER Flowchart Diagram


Initialize
Input Cosmic-Ray Damaged Image
Distribute Subimages
Laplacian Convolution
Generate Background Flux Model
4 iterations for optimal removal of cosmic rays from WFPC2 images
Yes
Another Iteration?
No
Combine Subimages
Output Clean Image
Finalize
Generate Noise Model
Divide Laplacian by Noise Model
Removal of Large Structures
Identify Candidate Cosmic Rays
Compare Candidate Cosmic Rays with Median Filter Image
Subtract Background & Smooth Compenent of Objects
Compare Cosmic Ray Flux with Object Flux
Selectively Discard Cosmic Rays
Grow Cosmic Ray Mask by One Pixel
Grow Cosmic Rays & Lower Deection Threshold
Determine Number of Cosmic Rays Detected
Create Clean Image
Key
Director Node
text
Actor Nodes
text

L.A.COSMIC algorithm:
van Dokkum, P. G. 2001, PASP, 113, 1420 – 1427

Mighell (HPEC 2010)

BAE Systems' RAD750 flight computer



Form factor
– CompactPCI 3U (100 mm x 160 mm)
– Weight: 549 grams

Memory
– 128 MB SDRAM, 256 kB SUROM

Radiation-hardness
– Total dose: >100 Krad (Si)
– SEU: 1.9 E-4 errors/card-day
(90% W. C. GEO) varies with orbit
– Latchup-immune

Performance
– >260 Dhrystone 2.1 MIPS @ 132 MHz
– 4.3 SPECint95 4.6 SPECfp95 at 132 MHz

Power supply
– 3.3V ± 10%
– (2.5V generated via on-board regulator)

Power dissipation
– <10.8W

Rail temperature range
– -55°C to +70°C

Pro:

- Radiation Hardened
- Low power (< 11 W)
- Proven platform (TRL 9)

Con:

- Slow (132 MHz)
- Single Core
- Expensive ($150,000++)

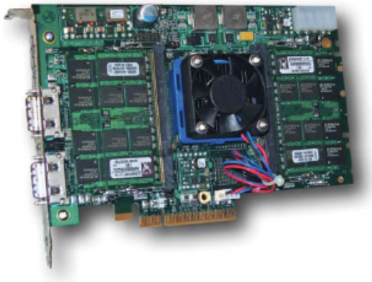
Credit: BAE Systems

Porting path to the 49-core Maestro processor:

✔ Beowulf cluster

↓

Tilera 700-MHz 64-core TILE64 processor



↓

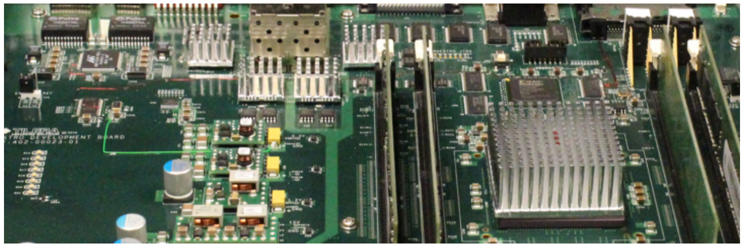
Tilera TILE64 processor simulator

↓

Maestro processor simulator

↓

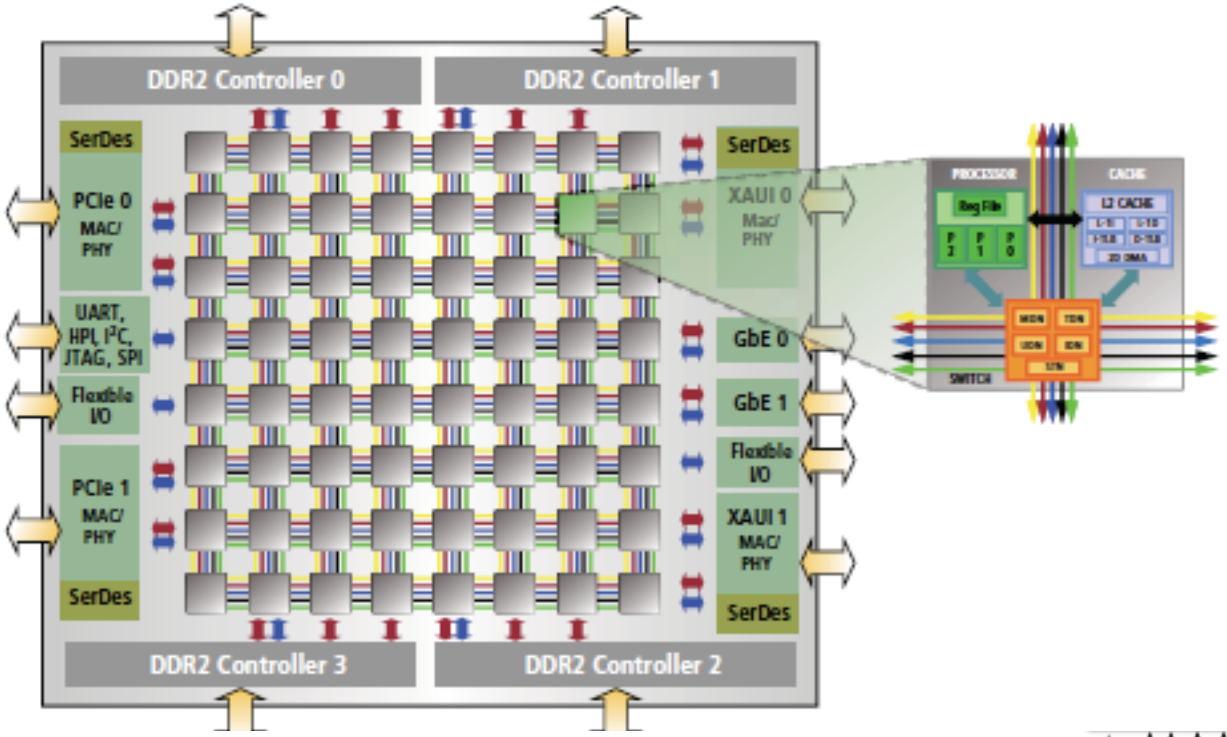
350-MHz Maestro 49-core processor



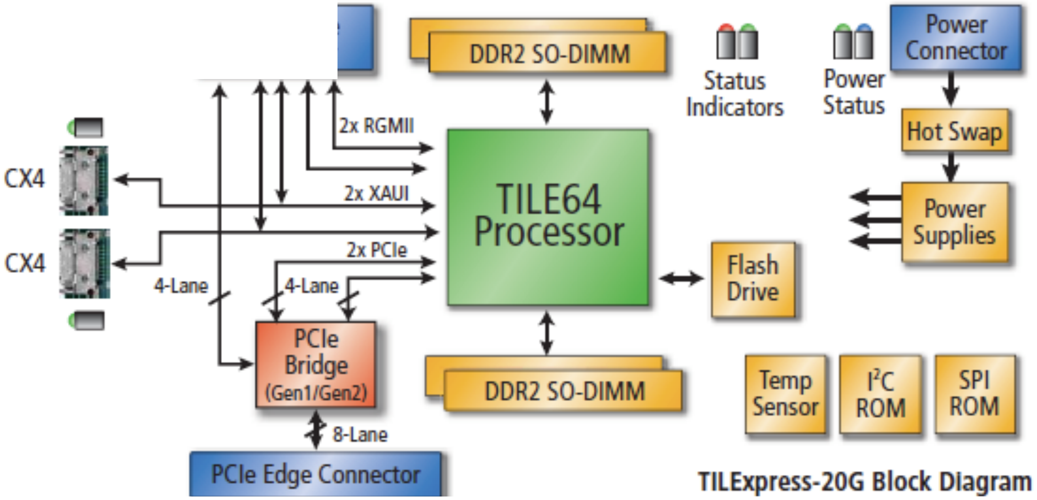
Hardware/Software solution

Software solution

Tilera's 64-core TILE64 processor



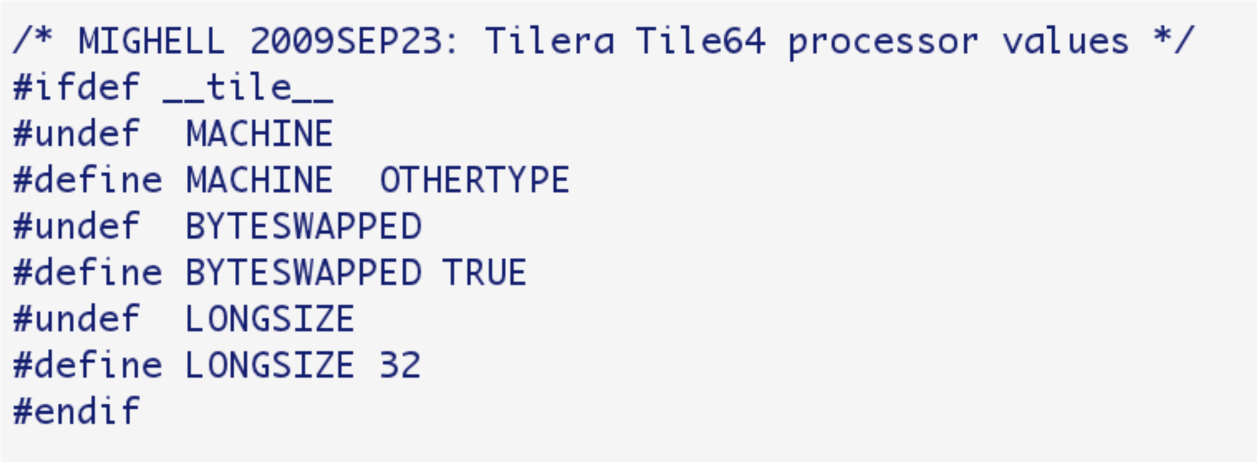
TILExpress-20G



TILExpress-20G Block Diagram

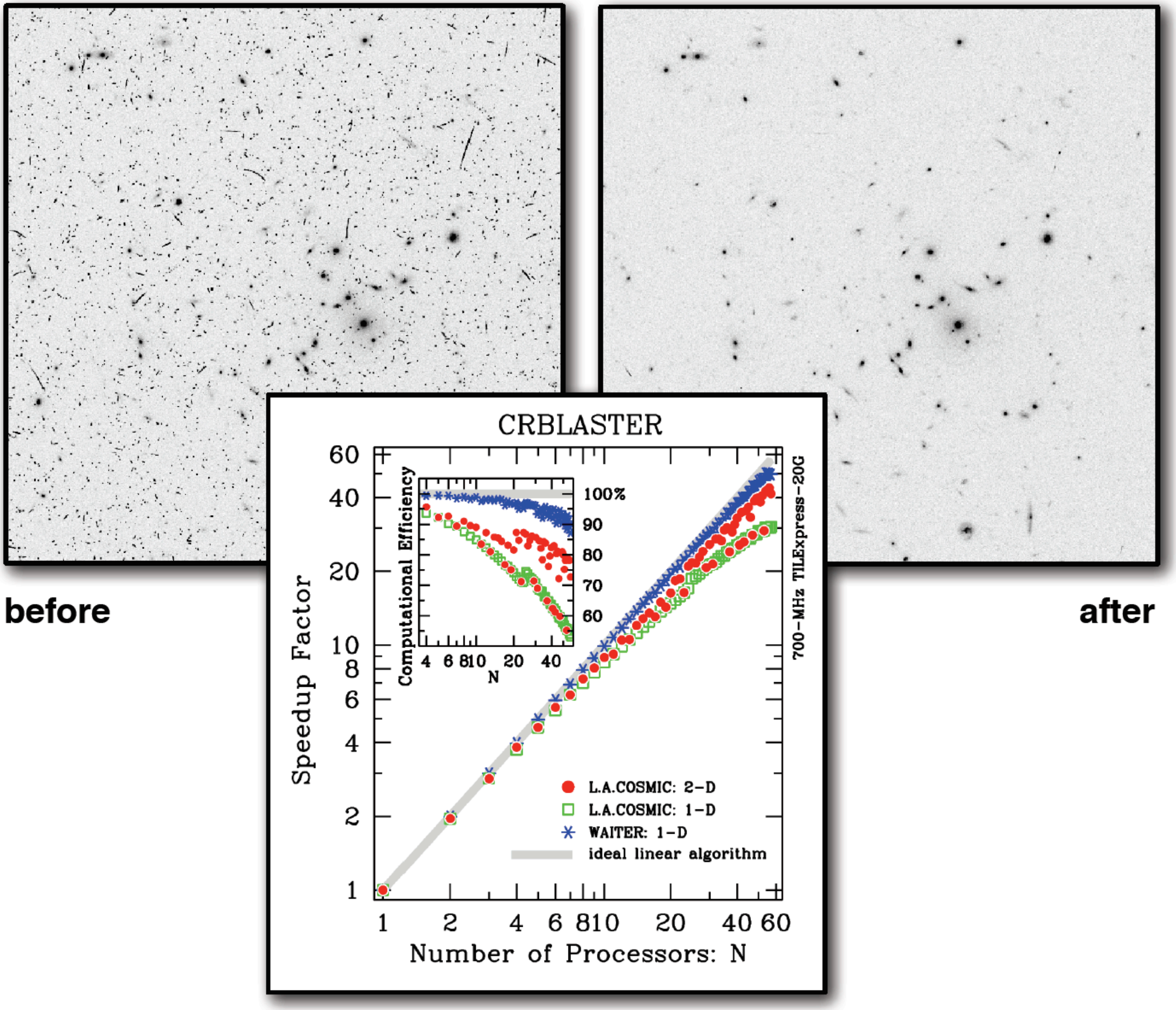
Credit: Tilera Corporation

CFITSIO: fitsio2.h needed to be modified:



```
/* MIGHELL 2009SEP23: Tilera Tile64 processor values */
#ifdef __tile__
#undef  MACHINE
#define MACHINE  OTHERTYPE
#undef  BYTESWAPPED
#define BYTESWAPPED TRUE
#undef  LONGSIZE
#define LONGSIZE 32
#endif
```

CRBLASTER then compiled and worked on the TILE64 platform without any further modification to the source code.



before

after

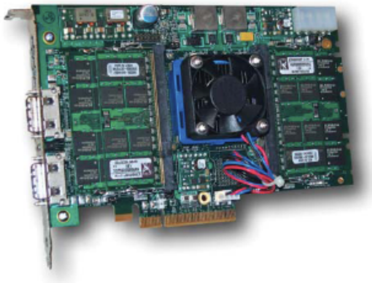
Mighell (Space Computing 2011)

Porting path to the 49-core Maestro processor:

✓ Beowulf cluster

↓

✓ Tilera 700-MHz 64-core TILE64 processor



↓

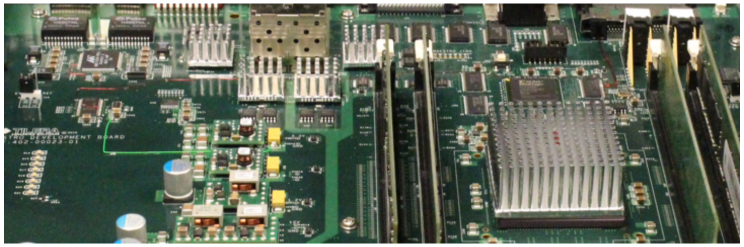
Tilera TILE64 processor simulator

↓

Maestro processor simulator

↓

350-MHz Maestro 49-core processor



Hardware/Software solution

Software solution

The software simulators are *slow* and can use only one core (tile).

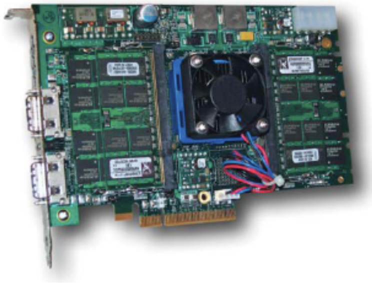
Use 50x50 images instead of 800x800 → 256 speedup. Runtimes now ~900 s instead of days…

Porting path to the 49-core Maestro processor:

✓ Beowulf cluster

↓

✓ Tilera 700-MHz 64-core TILE64 processor



↓

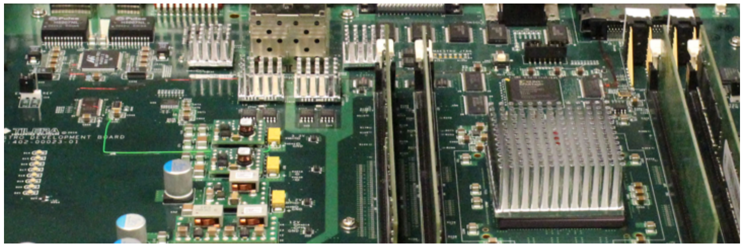
✓ Tilera TILE64 processor simulator

↓

Maestro processor simulator

↓

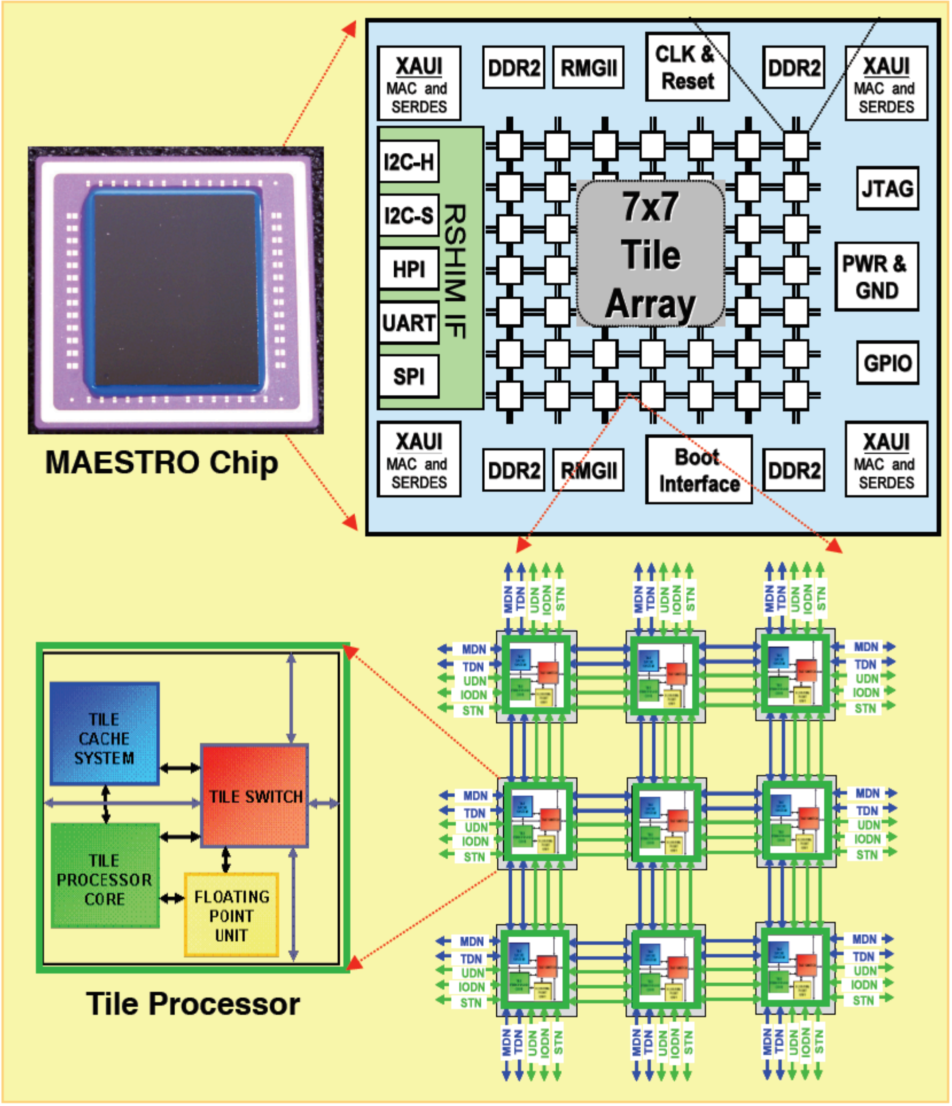
350-MHz Maestro 49-core processor



Hardware/Software solution

Software solution

49-core RHDB MAESTRO Processor



Courtesy of Boeing SSED (Solid-State Electronics Development)

The Maestro cross-compiler (tile-cc) is much closer to ANSI/ISO C standard than the Tilera C compiler which has many C99 features.

➔ Remove C99 features from code.

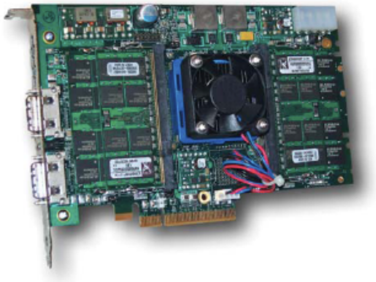
Porting path to the 49-core Maestro processor:

24 hours

✓ Beowulf cluster

↓

✓ Tilera 700-MHz 64-core TILE64 processor



↓

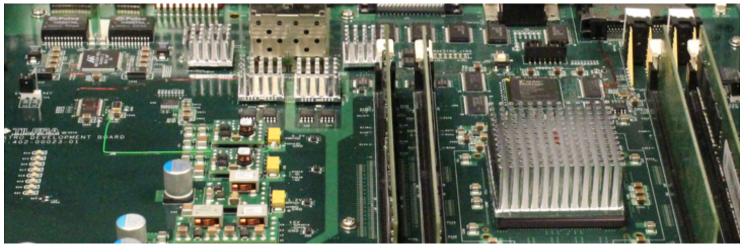
✓ Tilera TILE64 processor simulator

↓

✓ Maestro processor simulator

↓

350-MHz Maestro 49-core processor



An easy port:
24 hours spread over a few days.

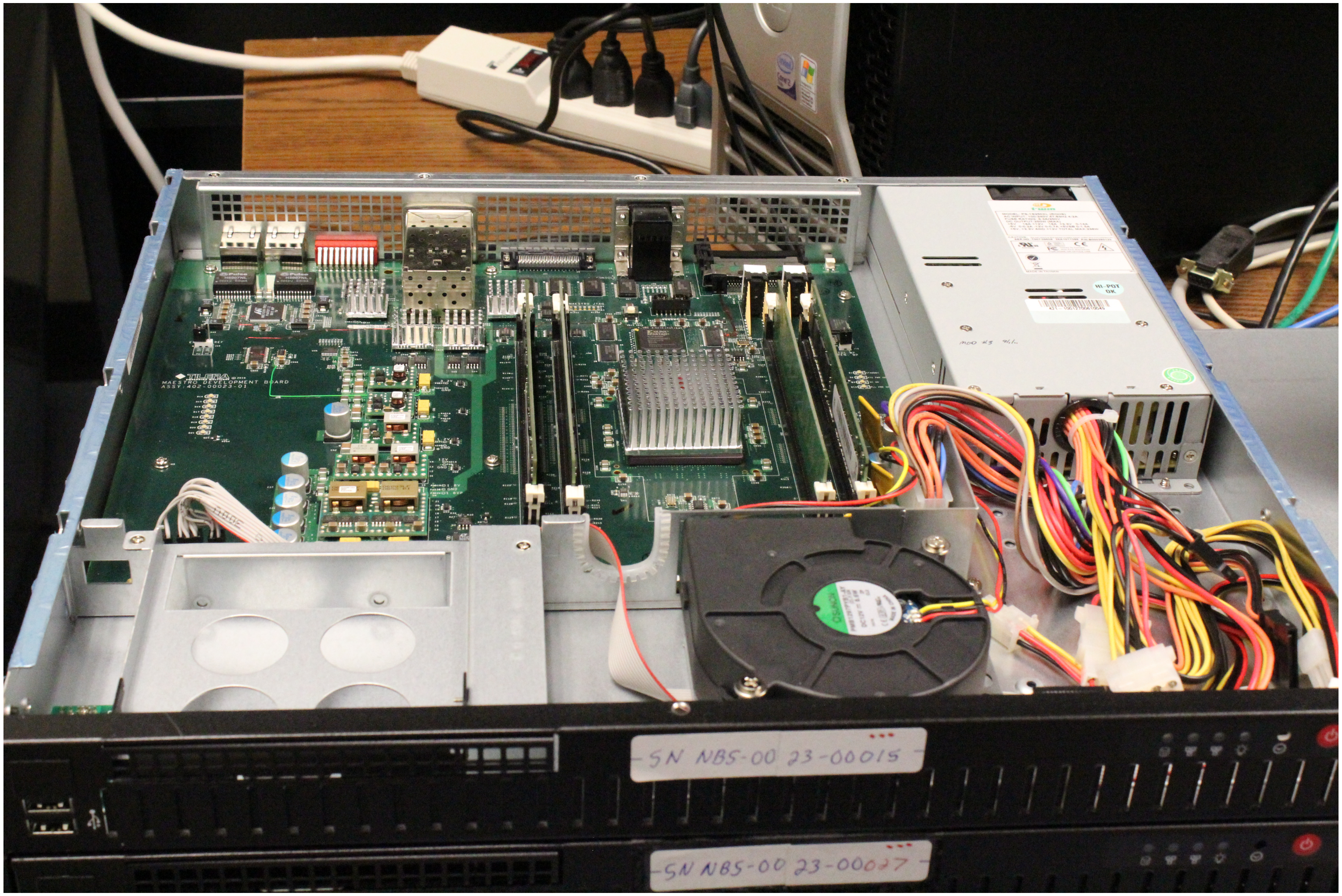
Hardware/Software solution

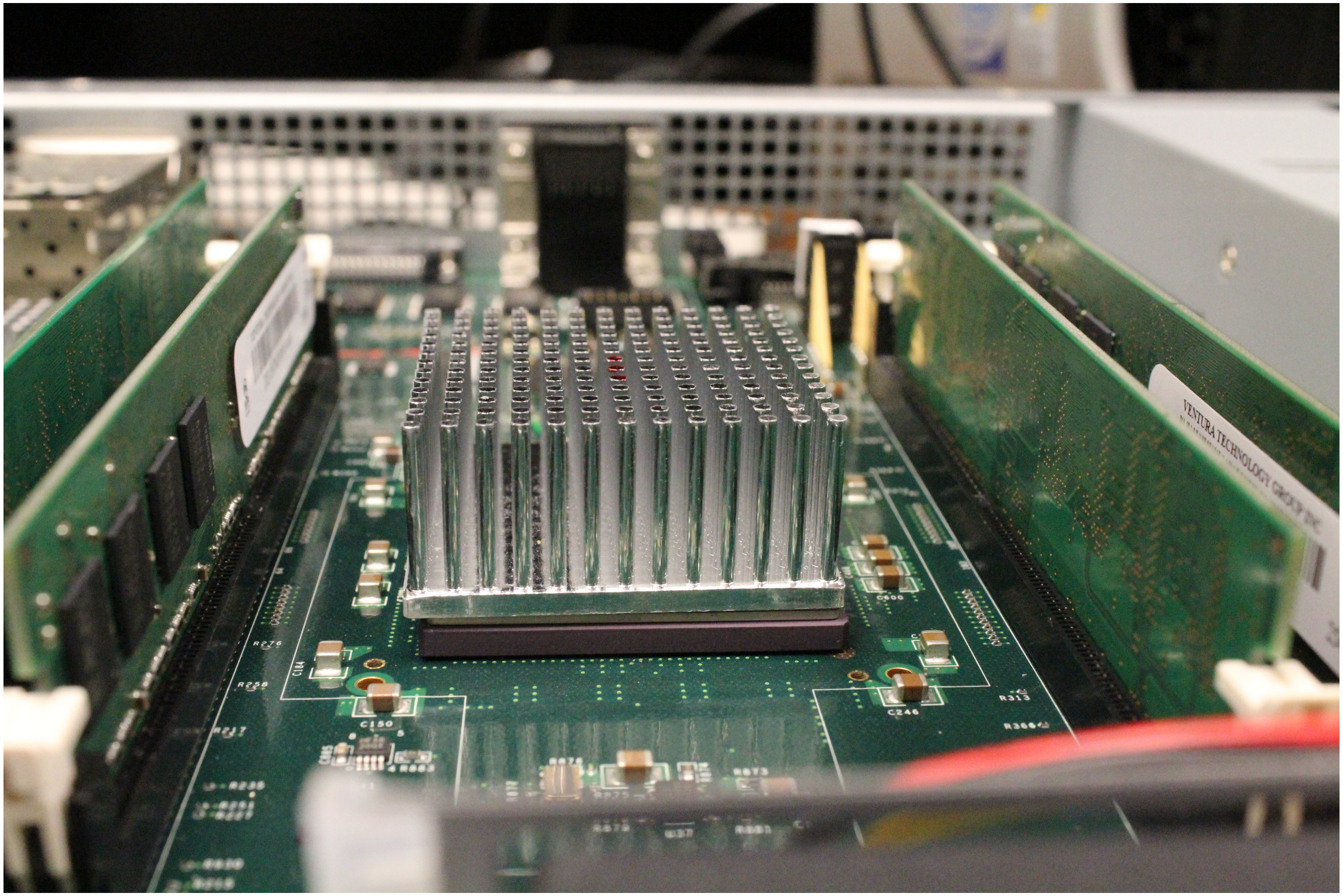
Software solution

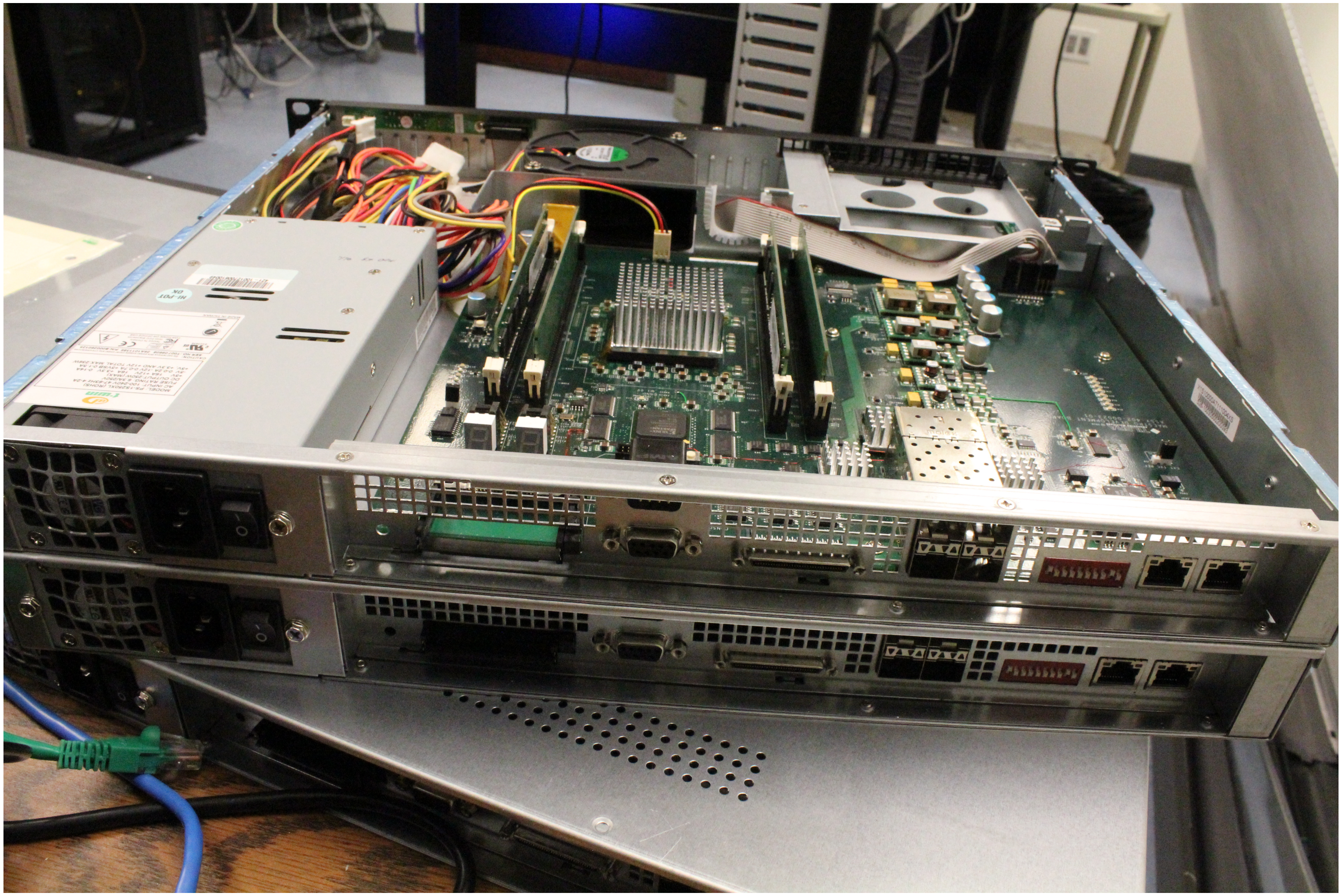
Wait many months…

for real hardware to become available for testing

MDB: Maestro Development Board (350 MHz)




VENTURA TECHNOLOGY GROUP INC
C150
C246
C184
R313
R276
C600



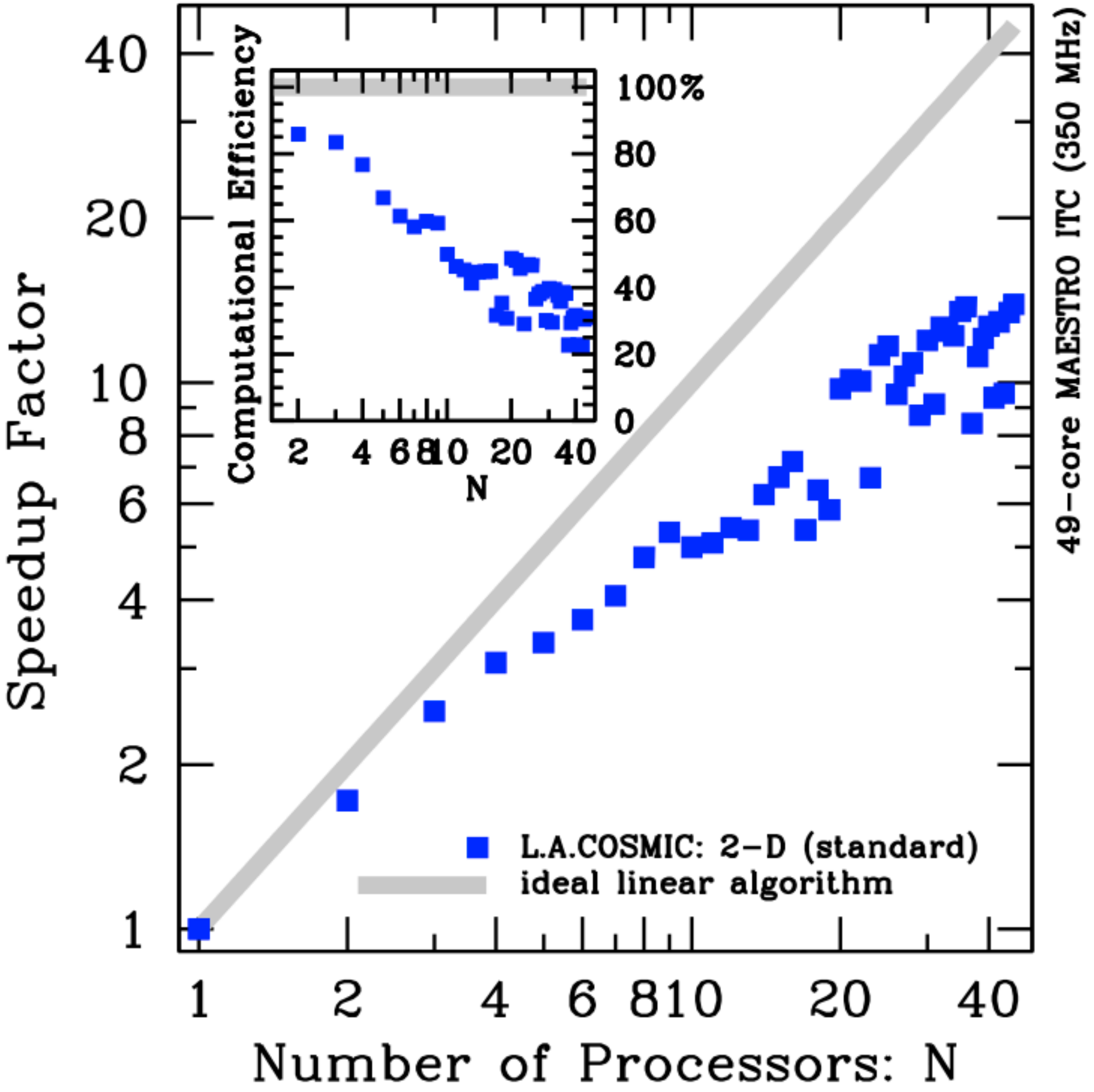
Challenges

- No Ethernet connectivity (being tested).
- 20+ minute bootrom upload process (via UART at 14,400 bytes per second).
- a *physical* reset is required to run a new bootrom.

Solutions

- Communicate using a text console via UART.
- Carefully plan testing runs.
  Bootrom images need to contain all required executable binaries along with all input images and all comparison (“gold standard”) output images.
- Use the remote reset website at ISI-East.
  (Check to make sure that nobody else is currently using the machine!)
- Be patient with 20+ minute bootrom uploads.

CRBLASTER



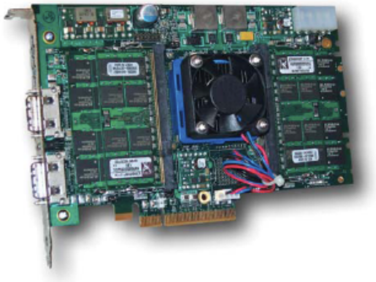
Speedup Factor
1
2
4
6
8
10
20
40
Number of Processors: N
1
2
4
6 8 10
20
40
49−core MAESTRO ITC (350 MHz)
Computational Efficiency
100%
80
60
40
20
0
2
4 6 8 10
20
40
N
L.A.COSMIC: 2−D (standard)
ideal linear algorithm

Porting path to the 49-core Maestro processor:

✔ Beowulf cluster

↓

✔ Tilera 700-MHz 64-core TILE64 processor



↓

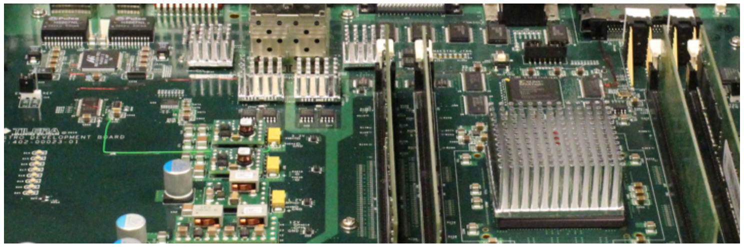
✔ Tilera TILE64 processor simulator

↓

✔ Maestro processor simulator

↓

✔ 350-MHz Maestro 49-core processor

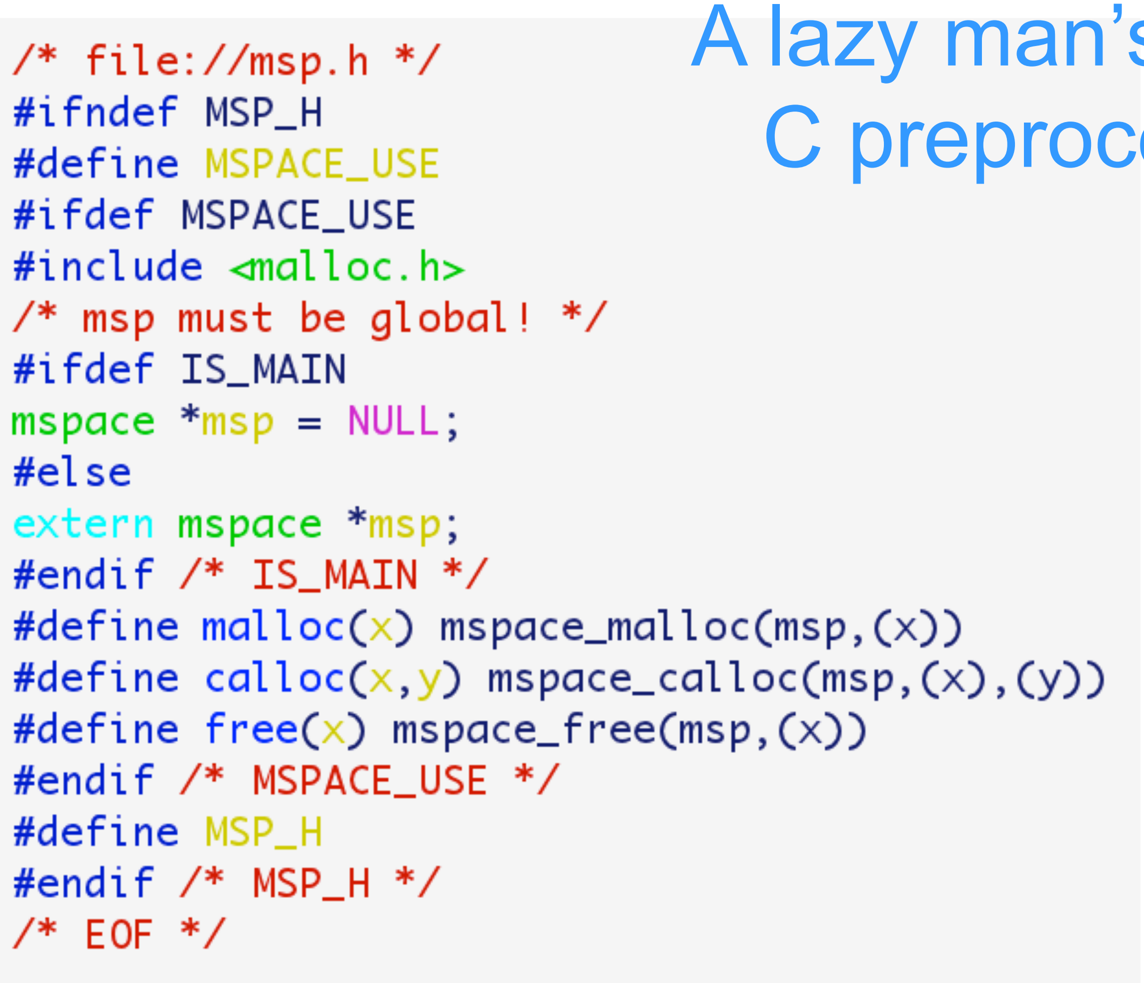


Hardware/Software solution

Software solution

Can we improve performance with user-defined memory controller allocation?

A lazy man’s solution: C preprocessor “abuse”

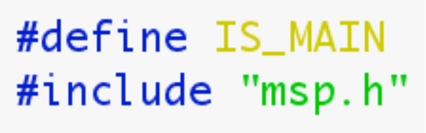


```
/* file://msp.h */
#ifndef MSP_H
#define MSPACE_USE
#ifdef MSPACE_USE
#include <malloc.h>
/* msp must be global! */
#ifdef IS_MAIN
mspace *msp = NULL;
#else
extern mspace *msp;
#endif /* IS_MAIN */
#define malloc(x) mspace_malloc(msp,(x))
#define calloc(x,y) mspace_calloc(msp,(x),(y))
#define free(x) mspace_free(msp,(x))
#endif /* MSPACE_USE */
#define MSP_H
#endif /* MSP_H */
/* EOF */
```

For further information:
Tilera Multicore Development Environment
Application Libraries Reference Manual
(Doc. No. UG227, 2010)

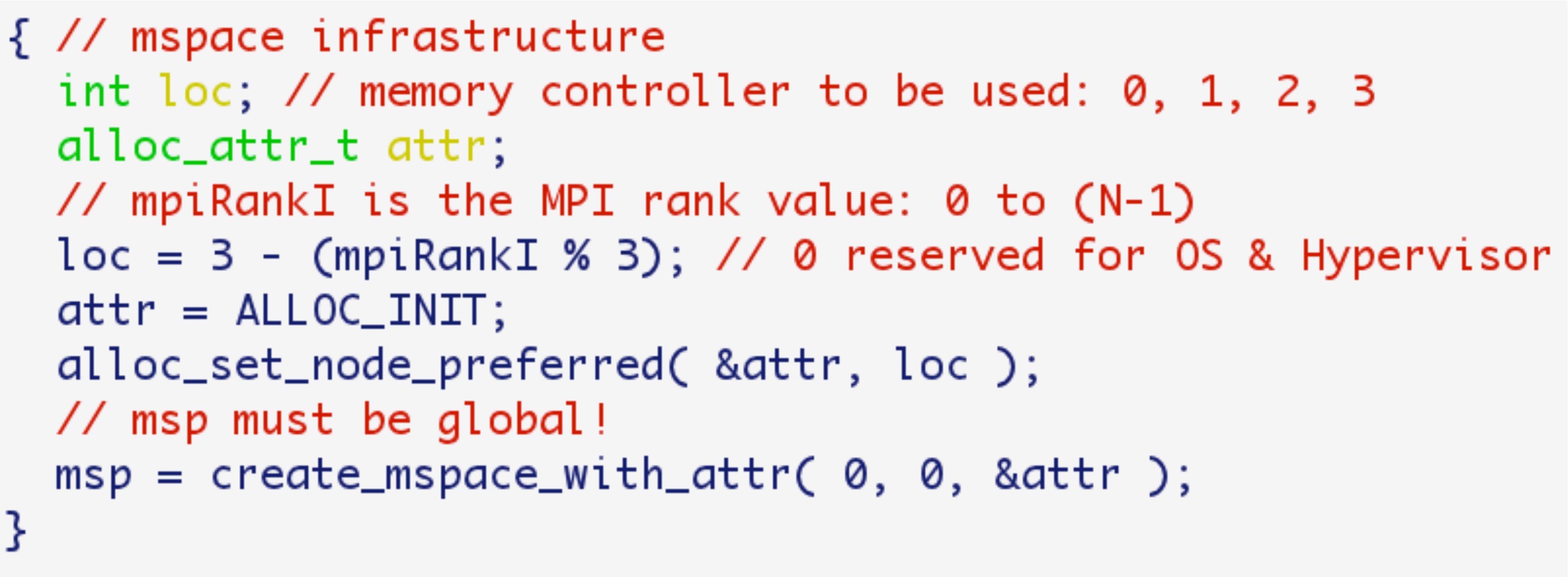
Tilera’s “mspace” code in malloc.h appears to based on an old version of Doug Lea’s malloc.c code (a.k.a. dlmalloc) at the website ftp://g.oswego.edu/pub/misc/malloc.c

main.c:



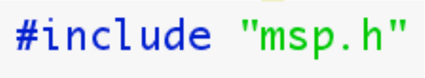
```
#define IS_MAIN
#include "msp.h"
```

<snip>



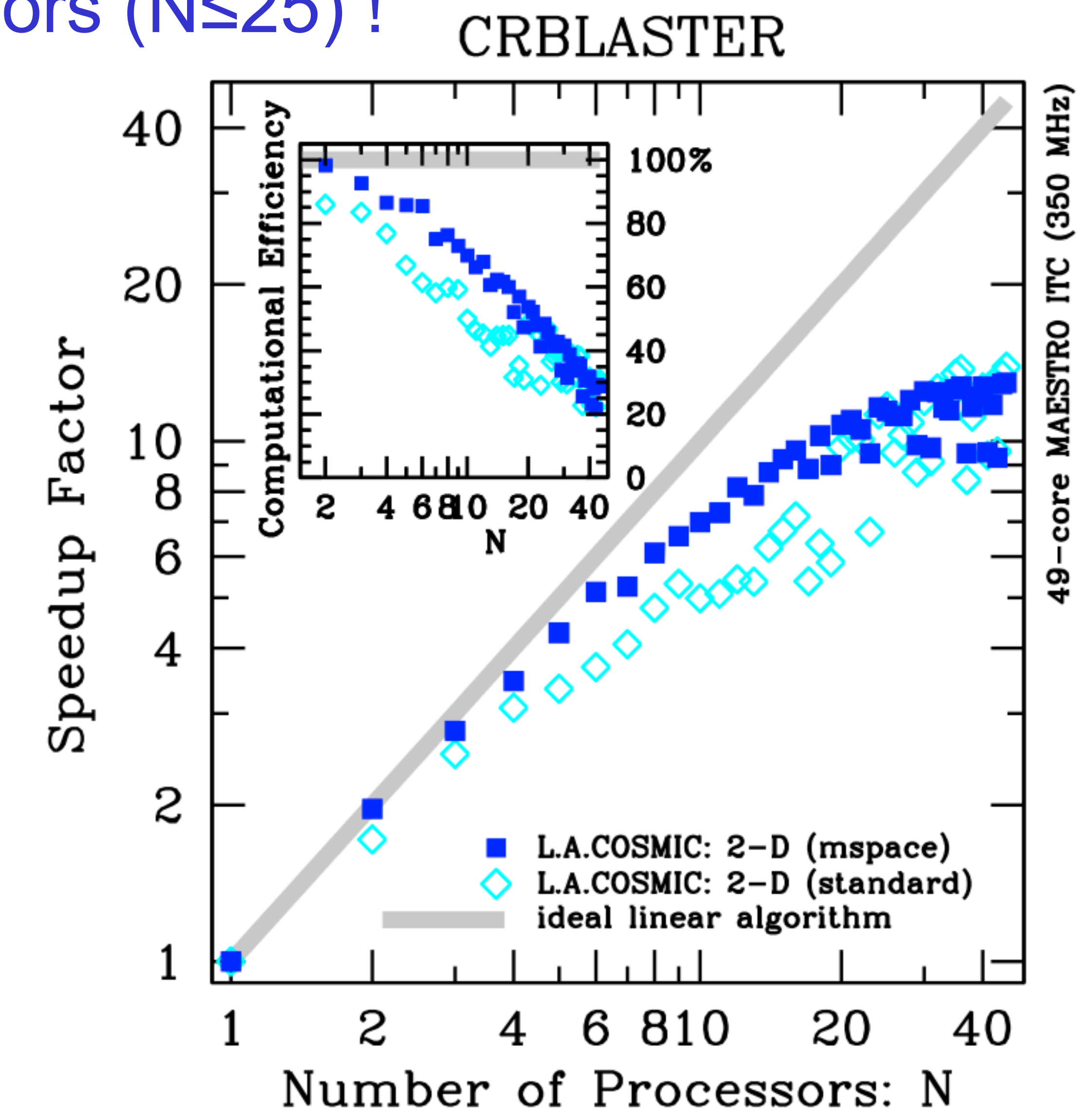
```
{ // mspace infrastructure
  int loc; // memory controller to be used: 0, 1, 2, 3
  alloc_attr_t attr;
  // mpiRankI is the MPI rank value: 0 to (N-1)
  loc = 3 - (mpiRankI % 3); // 0 reserved for OS & Hypervisor
  attr = ALLOC_INIT;
  alloc_set_node_preferred( &attr, loc );
  // msp must be global!
  msp = create_mspace_with_attr( 0, 0, &attr );
}
```

remainder of *.c source files:



```
#include "msp.h"
```

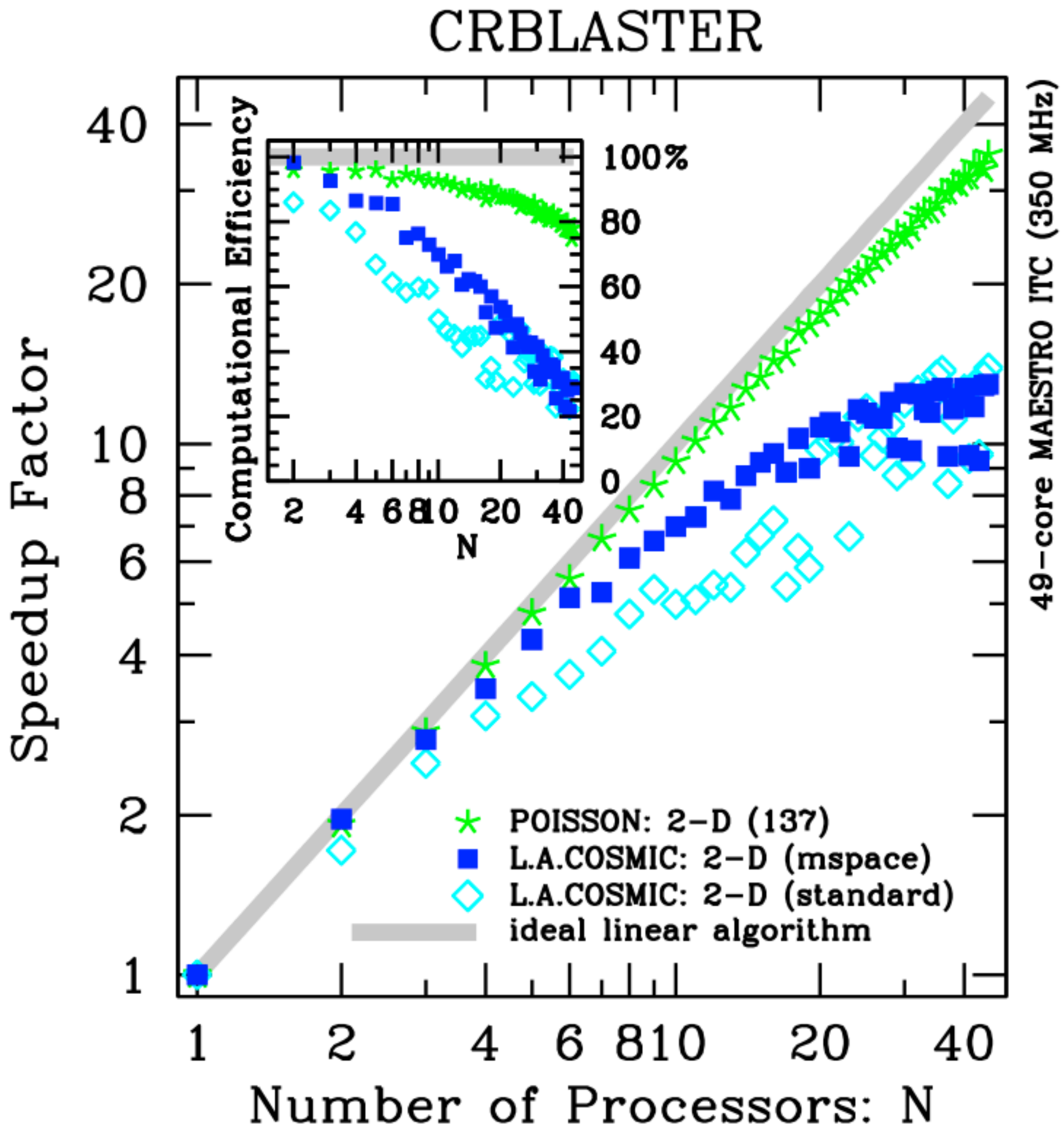
A significant improvement for a moderate number of processors (N≤25) !



Cosmic-ray rejection is a nonlinear application that is memory bound.

The CRBLASTER computational framework is actually much more efficient than previously shown.

Consider a computationally bound application…


CRBLASTER
Speedup Factor
1
2
4
6
8
10
20
40
Number of Processors: N
1
2
4
6
8
10
20
40
49-core MAESTRO ITC (350 MHz)
Computational Efficiency
100%
80
60
40
20
0
2
4
6
8
10
20
40
N
POISSON: 2-D (137)
L.A.COSMIC: 2-D (mspace)
L.A.COSMIC: 2-D (standard)
ideal linear algorithm

Serious problems arise when the ITC is used a lot…

```
==============================================

WARNING: Process 1405 terminated: Illegal instruction (4).

==============================================

(1,4) got double fault interrupt: PC 0xfd03_0000, ICS/PL 0x5

==============================================

Kernel took bad trap 2 at PC 0xfd23ef80
<snip>
Kernel panic - not syncing: Aiee, killing interrupt handler!

==============================================
```

Overheating was suspected…

Make sure the MDB has LOTS of cooling!

MDB➔

🡐 A/C unit

A heat pipe on top of the Maestro processor would be better!

The value of stress testing with real scientific applications: corner case exploration.

Conclusions

While it is still early days for the Maestro processor and Maestro Development Boards, we note that

- Maestro is easy to program.
  - With the MPI library, software developers can think of the Maestro processor as a Beowulf cluster on a chip.
- The CRBLASTER application was ported to an early Maestro Development Board (MDB) in 20 hours spread over several days.
- The CRBLASTER framework running the L.A.COSMIC algorithm on 36 tiles on the 350 MHz ITC had a speedup factor of 12.5 – giving the equivalent peak performance of a 4.3 GHz processor.
- The Maestro processor definitely has the potential to be an enabling technology for the next generation of U.S. Government satellites and NASA astrophysical missions.

This work was supported by two grants, Interagency Order No. NNG06EC81I and NASA Grant No. NNX10AD45G, from the **Applied Information Systems Research (AISR)** Program and the **Astrophysics Data Analysis Program (ADAP)**, respectively, of the Science Mission Directorate of the National Aeronautics and Space Administration.