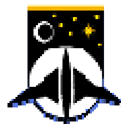
©1999 University of Applied Sciences Wolfenbuettel and CLE, Department of High Performance Computing

# A Test Suite for High-Performance Parallel Java

Jochem Häuser, Thorsten Ludewig, Roy D. Williams, Ralf Winkelmann,
Torsten Gollnick, Sharon Brunett, Jean Muylaert



presented at

5th National Symposium
on Large-Scale Analysis, Design and Intelligent Synthesis Environments

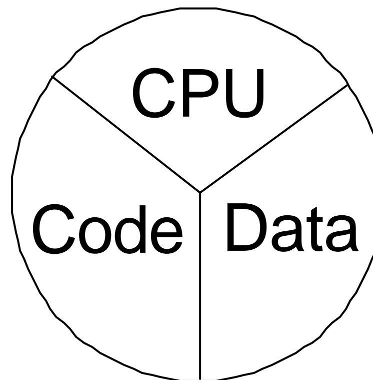
Williamsburg, VA, October 12th to 15th, 1999

# Introduction and Motivation

- Why we like to use Java for writing high-quality portable parallel programs?
  - pure object formulation (i.e. an object representation of a wing, fuselage, engine etc. described by a set of classes containing the data structures and methods for a specific item)
  - strong typing
  - exception model
  - elegant threading
  - portability

# What is a Thread?

- Virtual CPU
- Three parts of a Thread:
  - CPU
  - Code
  - Data




A thread or execution context

# Why Threads are good for CFD

- Threads as a general parallelization strategy for CFD codes
- Advanced numerical schemes in CFD, i.e. GMRES, do not require the same computational work for each grid cell.
- Sophisticated dynamic load balancing algorithms on distributed-memory machines

LOGISTIKZENTRUM CLE

# Java Numerical Performance

- "write-once-run-anywhere"
- porting Java is certainly easier than porting C, C++, or Fortran codes
- portability is obtained because Java is generally not compiled to a platform-specific executable, but converted to so-called *byte-code*
- have a serious performance impact
- Hewlett-Packard, IBM, Sun and Tower Technology, offer (or will offer) Java compilers, that create native code directly
  - Thus a user has the choice of portability or performance

# Java Numerical Performance

- garbage-collector thread is always running
  - makes Java slower than C
  - provides an enormous boost to the programmer's productivity
- Multithreading
  - a direct parallelization strategy
  - in C it is more difficult to manage threads
- Runtime checking
  - null reference, array bounds, ...
  - makes Java programs much more robust

# JParNSS

- We are in the process of developing a parallel Navier-Stokes flow solver ***JParNSS***, using the principles learned from this test suite.
- Thus, the test suite presented here concentrates on two performance aspects:
  - single-node performance of Java
  - speedup (ratio of single-node speed to multiple-node speed)

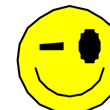
LOGISTIKZENTRUM CLE

# The Test Machines

- HP V-class
  - 32 PA-8200 processors running at 240 MHz, with 16 GByte RAM. Each processors has 2 MByte instruction as well as data cache
- Sun Enterprise E450
  - SMP architecture, powered by four 300 MHz UltraSPARC II processors which are connected by a 1.6-GB/s UPA interconnect to 1.7 GB of shared memory
- no name PC
  - a Pentium II with 300MHz running Linux

# The Test Suite

- Square Root (parallel efficiency, thread scheduling)
- Matrix Multiply (comparison of Java and C code)
- Mandelbrot (thread scheduling and dynamic load balancing)
- Laplace (domain decomposition and synchronization strategies)
- JParFw - Java Parallel Framework (client-server concept)
- JParEuler (template for a flow solver)

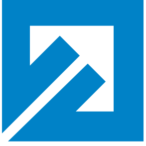


# Square Root

- many identical threads are used for simple arithmetic

Computing times and parallel speedups for square root program on HP V-Class

| number of threads | time in s | parallel speedup |
|---|---|---|
| 1 | 12:41 | 1 |
| 2 | 6:34 | 1,93 |
| 3 | 4:20 | 2,92 |
| 4 | 3:17 | 3,86 |
| 8 | 1:39 | 7,69 |
| 9 | 1:28 | 8,65 |
| 16 | 0:50 | 15,22 |
| 32 | 0:26 | 29,27 |

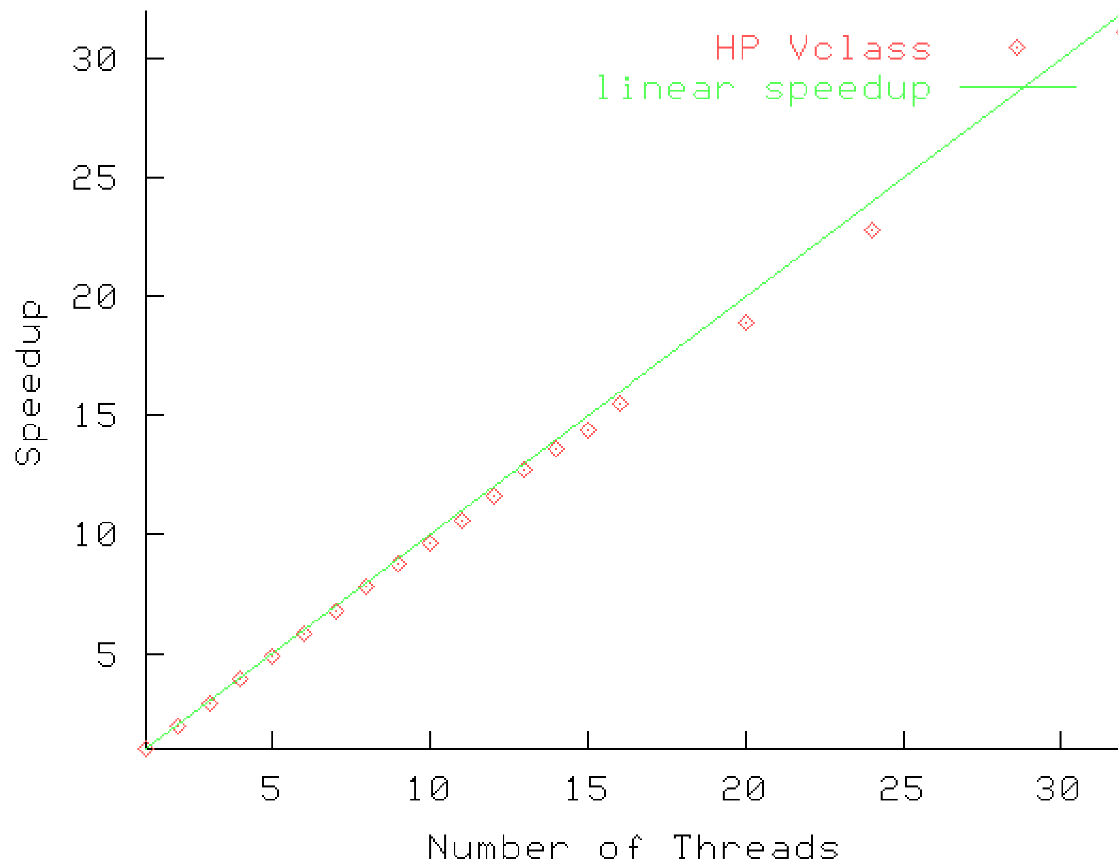




LOGISTIKZENTRUM CLE

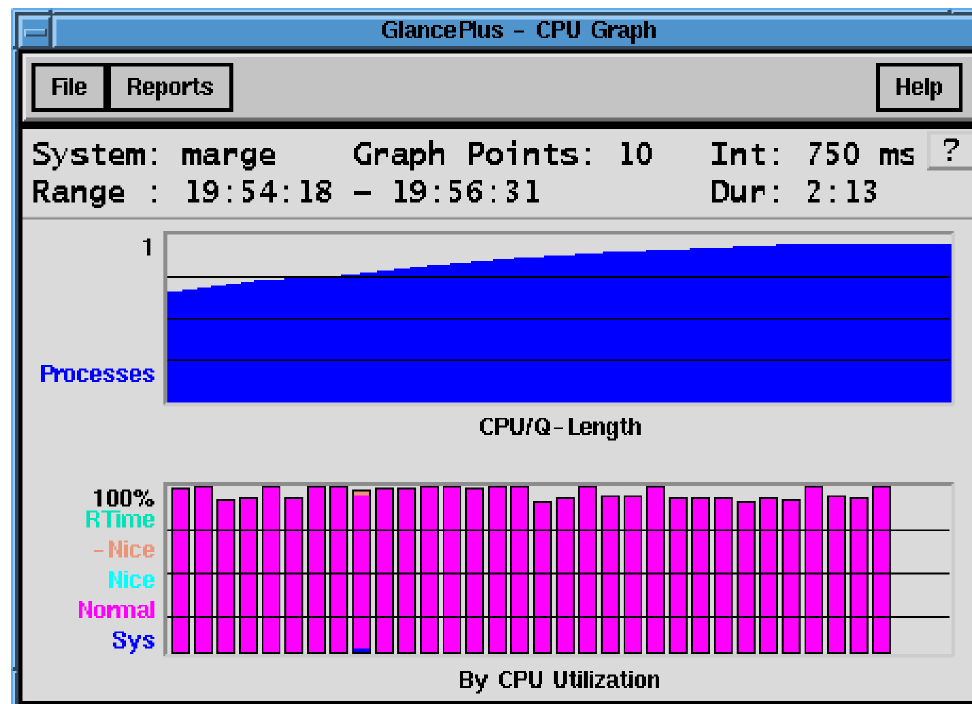
# Square Root

Computing times and parallel speedups for square root program on HP V-Class



# Square Root

The program GlancePlus is used on the HP V-Class to illustrate the parallel runtime behavior for the square root example.



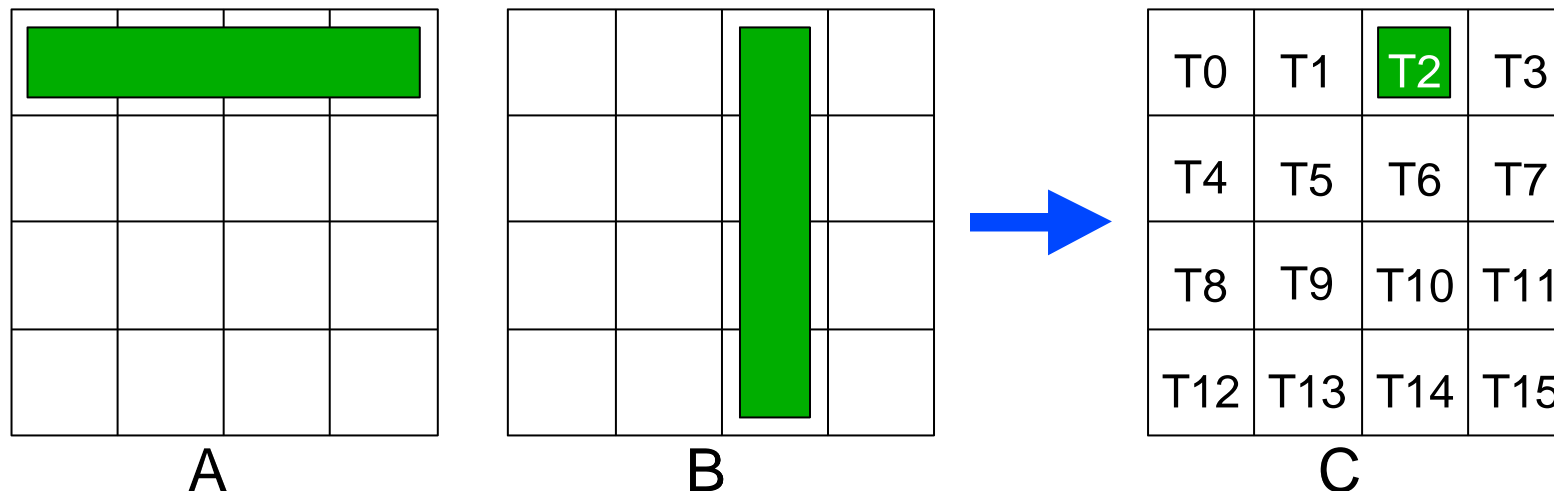

history of the averaged machine load

CPU utilization

LOGISTIKZENTRUM CLE

# Matrix Multiply

- Parallel matrix multiplication is implemented by block matrices, as shown in the Figure Matrices **A** and **B** are multiplied to produce **C**




**The multi-threaded matrix multiplication is performed by splitting matrix C into partitions. Each partition is then calculated by one thread, with the thread numbering as shown for matrix C. Concurrent access to the memory containing A and B is necessary: here we see the memory that thread 2 accesses.**

# Matrix Multiply

- for comparison, we have a Java and a C-coded version of the sequential block-matrix multiply that does not use threads
  - to compare floating point performance for scientific applications between C and Java on the test machines
  - to measure parallel efficiency of a multithreaded application

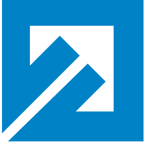
©1999 University of Applied Sciences Wolfenbuettel and CLE, Department of High Performance Computing



# Sequential Matrix Multiplication

| Hardware and Software specifications | MFlops per second for dirfferent matrix sizes | | |
|---|---|---|---|
| | 30x30 | 100x100 | 300x300 |
| HP Vclass, C-code | 242,00 | 237,00 | 114,00 |
| HP Vclass, Java 1.1.7 | 9,33 | 9,57 | 9,54 |
| Sun E450, C-code | 176,86 | 157,73 | 35,24 |
| Sun E450, Java 1.1.7 | 6,35 | 6,72 | 5,87 |
| Sun E450, Java 1.2 | 17,08 | 12,65 | 8,90 |
| Pentium, C-Code | 90,00 | 91,74 | 39,82 |
| Pentium, Java IBM 1.1.6 | 24,80 | 22,79 | 11,21 |

The performance, in megaflops, of the sequential matrix-multiply program on the one processor of the HP-Vclass, one processor of the Sun E450, and a Pentium II PC running Linux.



# Multi-threaded Matrix Multiplication

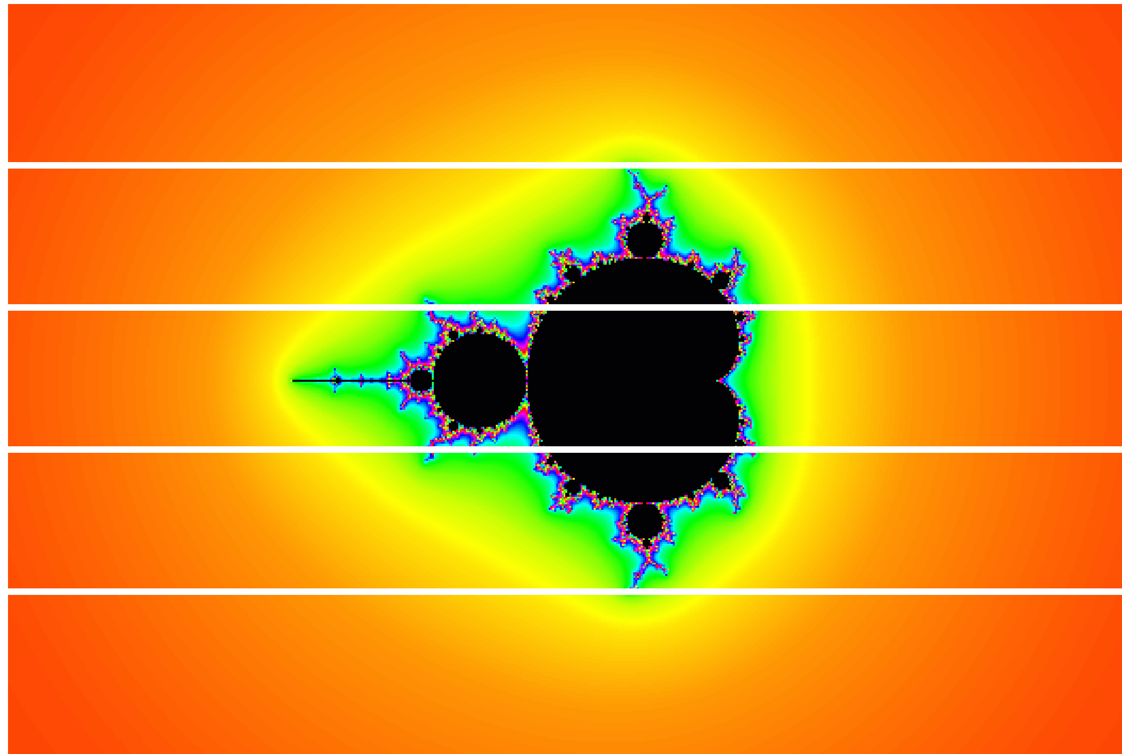
Megaflop rates for the pure Java multithreaded matrix-multiply benchmark.

| number of threads | HP using 16 CPUs | | HP using 1 CPU | | Sun using 4 CPUs | |
|---|---|---|---|---|---|---|
| | 30x30 | 300x300 | 30x30 | 300x300 | 30x30 | 300x300 |
| 1 | 7,01 | 8,64 | 6,51 | 8,66 | 13,40 | 9,06 |
| 4 | 11,38 | 33,49 | 3,86 | 8,68 | 19,21 | 23,56 |
| 9 | 6,33 | 72,53 | 2,40 | 8,73 | 12,25 | 22,00 |
| 16 | | 118,68 | | 8,69 | | 28,13 |
| 25 | 2,62 | 112,97 | 1,04 | 8,65 | 5,14 | 27,84 |
| 36 | 1,83 | 110,66 | 0,75 | 8,64 | 3,75 | 30,07 |
| 100 | 0,64 | 109,53 | 0,29 | 8,44 | 1,57 | 33,93 |

On the HP architecture a maximal speedup of 13,74 using 16 processors for the 300x300 matrix example was measured.

# Mandelbrot

- This code tests the self-scheduling of threads
- Computation of the Mandelbrot set performed on a 2D grid in the complex plane



# Mandelbrot

- Almost 100% parallel efficiency was achieved on the 4 processor Sun E450.
  - picture dimensions 3000 x 2000 pixels
  - 256 Threads
  - 5000 Iterations
- Problems with thread scheduling occurred on the HP.

LOGISTIKZENTRUM CLE

# Laplace Solver

- In this test program we are using a global thread synchronization instead of a loose thread synchronization.

| number of processors | time in s | parallel speedup |
|---|---|---|
| 1 | 893 | 1,00 |
| 2 | 639 | 1,40 |
| 3 | 562 | 1,59 |
| 4 | 578 | 1,54 |

Computing times and speedups on Sun E450 using 192 threads (16x12 blocks with 9600 cells) running 5000 iterations. We achieve in this test case 50% CPU load only.

# Laplace Solver

- with loose thread synchronization (a thread is blocked until it's neighboring threads become ready)

| number of processors | time in s | parallel speedup |
|---|---|---|
| 1 | 842 | 1,00 |
| 2 | 430 | 1,96 |
| 3 | 295 | 2,85 |
| 4 | 226 | 3,72 |

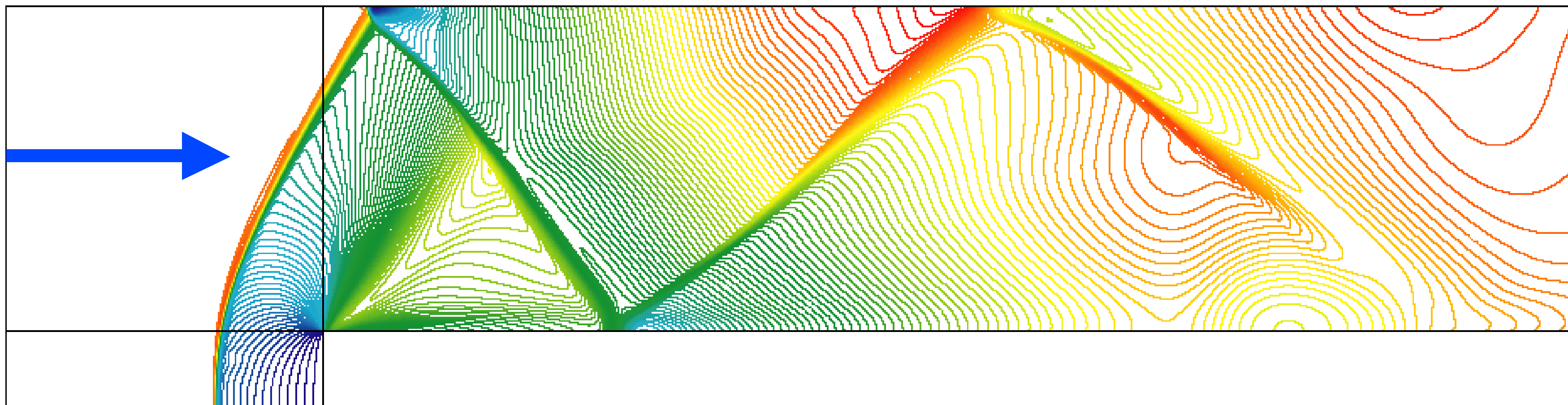
Computing times and speedups on Sun E450 using 192 threads (16x12 blocks with 9600 cells per block) running 5000 iterations.

# JParFw - Java Parallel Framework

- A complete pure Java parallel framework for solving problems in science and engineering.
- It is based on the client-server concept (Java RMI)
- Allows to perform a parallel computation using the internet
- The JParFw comprises three parts:
  - client (uses the server interface and implements the numerics)
  - server (server implementation)
  - share (server interface)

# JParEuler

- Is an Euler solver plugin for the JParFw
- Testcase is a Mach 3 Euler flow past a forward facing-step
- The simulations are based on structured multi-block grids



The computation is explicit and first order accurate. Shown is the Mach-number distribution.

# JParEuler

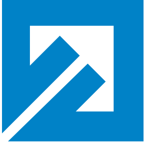
| number of blocks | number of cells | time in seconds | | parallel speedup |
|---|---|---|---|---|
| | | single processor | multi processor (16) | |
| 16 | 121104 | 3246,73 | 541,13 | 6,00 |
| 16 | 200704 | 6908,88 | 1077,20 | 6,41 |
| 16 | 484416 | 12905,88 | 2720,48 | 4,74 |
| 48 | 118803 | 2980,93 | 225,76 | 13,20 |
| 48 | 202800 | 5190,54 | 436,09 | 11,90 |
| 48 | 480000 | 12663,30 | 1162,54 | 10,89 |

HP V-Class times for 320 iterations

# JParEuler

| number of blocks | number of cells | time in seconds | | parallel speedup |
|---|---|---|---|---|
| | | single processor | multi processor (4) | |
| 16 | 121104 | 852,79 | 258,26 | 3,30 |
| 16 | 200704 | 1571,76 | 532,74 | 2,95 |
| 16 | 484416 | 4277,96 | 1593,45 | 2,68 |
| 48 | 118803 | 756,24 | 195,95 | 3,86 |
| 48 | 202800 | 1409,96 | 378,61 | 3,72 |
| 48 | 480000 | 3892,67 | 1077,06 | 3,61 |

Sun E450 times for 320 iterations





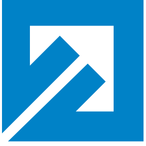


# Comparing Solaris 2.6 and Solaris 7

- JparEuler is used to demonstrate the influence of the operating system on the parallel performance while using the same JDK release (build Solaris_JDK_1.2.1_03, native threads, sunwjit)

| number of blocks | number of cells | Operating System | time in s multiprocessor (4) |
|---|---|---|---|
| 48 | 42000 | Solaris 2.6 | 282,04 |
| 48 | 42000 | Solaris 7 | 258,16 |

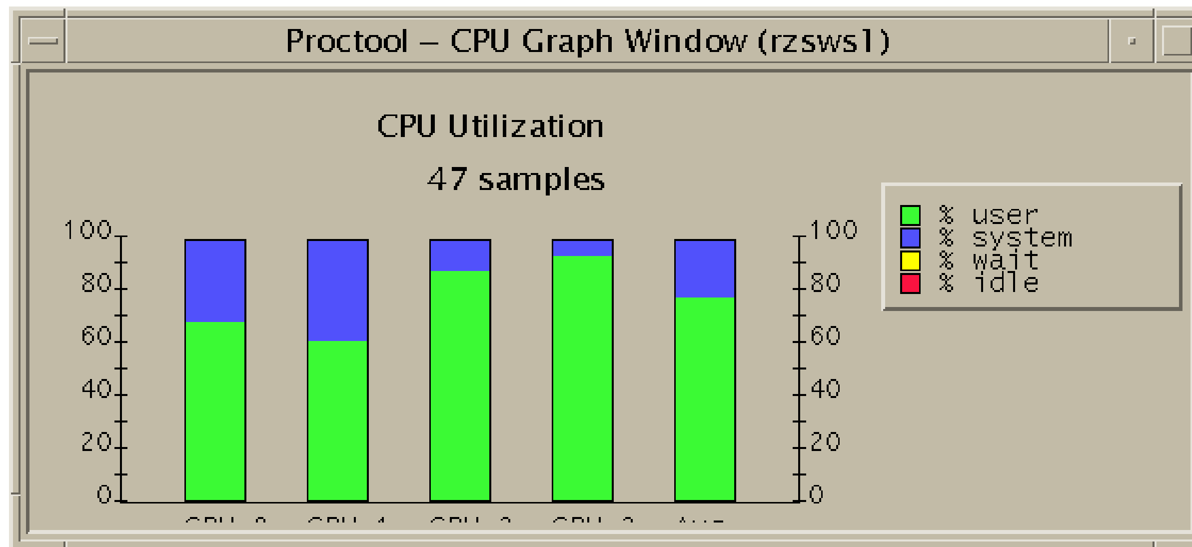
JParEuler on Sun E450, 1000 iterations



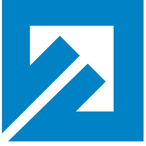


LOGISTIKZENTRUM CLE

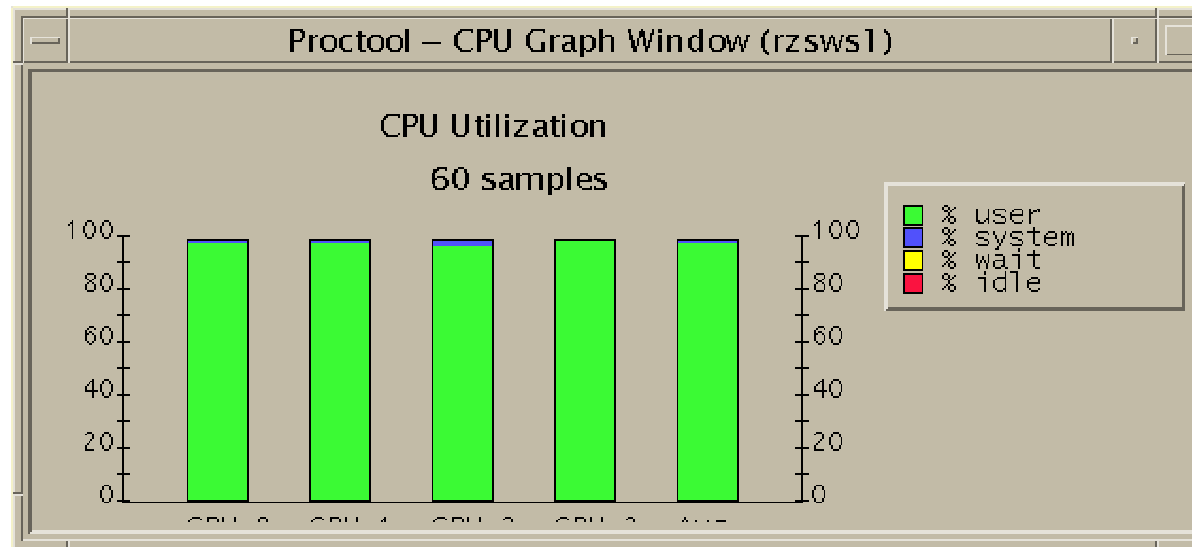
# Comparing Solaris 2.6 and Solaris 7




Snapshot of CPU utilization during the first minute of the JParEuler computation on the Sun E450 using Solaris 2.6. The system requires about 20% of the CPU resources for internal Thread (LWP) handling.



# Comparing Solaris 2.6 and Solaris 7




Snapshot of CPU Ultilization after the OS Thread handling is optimized. (JParEuler computation on Sun E450 using Solaris 2.6.)

LOGISTIKZENTRUM CLE

# Conclusions

- The parallel efficiency is obtained if a sufficient number of threads and sufficient computational work within a thread can be provided.
- With the scientific and engineering problems that we have in mind, these requirements are easily satisfied.
- Today, the speed of the Java code generated by the compilers of the main hardware vendors is unsatisfactory.
- Substantial speedups can be expected within the next 18 month.
- The IBM alphaWorks compiler delivered good results.

# Conclusions

- The new Solaris release has shown that the OS itself can decrease computing time by more efficient thread handling.
- We are therefore not concerned by the speed issue, leaving this problem to the compiler builders.
- Further work will be needed, but we following Kernighan's rules *Make it right before you make it faster* as well *Don't patch bad code, rewrite it*, the latter rule being the reason for a pure Java flow solver code.

LOGISTIKZENTRUM CLE

# Acknowledgements

- The *Test Suite for High-Performance Parallel Java* is part of the *JavaPar* project which was partly funded by the ministry of Sciences and Culture of the State of Lower Saxony, Germany and the European Commision under the contract *JavaPar* 1998.262.

- We would like to thank the Center for Advanced Computing Research at Caltech for hosting Ralf Winkelmann while this work was completed.

- This work contains parts of the Ph.D. work of Thorsten Ludewig

LOGISTIKZENTRUM CLE

# References


- Eiseman, Peter R., GridPro v3.1, The CFD Link to Design, Topology Input Language Manual, Program Development Corporation Inc., 300 Hamilton Ave., Suite 409, White Plains, NY 10601, 1998.
- Fox, G.C. (ed.), Java for Computational Science and Engineering- Simulation and Modeling I, Concurrency Practice and Experience, Vol. 9(11), June 1997,Wiley.
- Fox, G.C. (ed.), Java for Computational Science and Engineering- Simulation and Modeling II, Concurrency Practice and Experience, Vol. 9(11), November 1997,Wiley.
- James Gosling, Henry McGilton, The Java Language Environment - A White Paper, Sun Microsystems October 1995, http://www.javasoft.com/docs/.
- Häuser J., Williams R.D, Spel M., Muylaert J., ParNSS: An Efficient Parallel Navier-Stokes Solver for Complex Geometries, AIAA 94-2263, AIAA 25th Fluid Dynamics Conference, Colorado Springs, June 1994.
- Häuser, J., Xia, Y., Muylaert, J., Spel, M., Structured Surface Definition and Grid Generation for Complex Aerospace Configurations, In: Proceedings of the 13th AIAA Computational Fluid Dynamics Conference Open Forum, June 29 - July 2, 1997, Part 2, pp. 836-837, ISBN 1-56347-233-3.
- Häuser J., Williams R.D., Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines, Int. Journal for Numerical Methods in Fluids 15,51-58. , John Wiley & Sons, June 1992.
- Häuser, J., Ludewig, Th., Gollnick, T., Winkelmann, R., Williams, R., D., Muylaert, J., Spel, M., A Pure Java Parallel Flow Solver, 37th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 99-0549 Reno, NV, USA, 11-14 January 1999.
- Winkelmann, R., Häuser J., Williams R.D, Strategies for Parallel and Numerical Scalability of Large CFD Codes, Comput. Methods Appl. Mech. Engrg. 174 (1999) 433-456, 1999.