Impact of OpenMP Optimizations for the MGCG Method

Osamu Tatebe, Mitsuhisa Sato,
Satoshi Sekiguchi

Electrotechnical Laboratory, RWCP

What is MGCG?

- It is the multigrid preconditioned conjugate gradient method.
- It is an efficient (parallel) Poisson solver even with severe coefficient jumps.

MGCG Method

Let $\mathbf{x}_0$ be an initial approximation.

$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$; $\bar{\mathbf{p}}_0 = \bar{\mathbf{r}}_0 = \mathbf{M}^{-1}\mathbf{r}_0$

for (i = 0; 1; $i = i + 1$) {

$\alpha_i = (\bar{\mathbf{p}}_i, \mathbf{r}_i) / (\bar{\mathbf{p}}_i, \mathbf{A}\bar{\mathbf{p}}_i)$

$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \bar{\mathbf{p}}_i$

$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\bar{\mathbf{p}}_i$

if (convergence) break;

$\bar{\mathbf{r}}_{i+1} = \mathbf{M}^{-1}\mathbf{r}_{i+1}$ // MG preconditioning

$\beta_i = (\bar{\mathbf{r}}_{i+1}, \mathbf{A}\bar{\mathbf{p}}_i) / (\bar{\mathbf{p}}_i, \mathbf{A}\bar{\mathbf{p}}_i)$

$\bar{\mathbf{p}}_{i+1} = \bar{\mathbf{r}}_{i+1} + \beta_i \bar{\mathbf{p}}_i$

}

Preconditioning Matrix

Multigrid Preconditioner

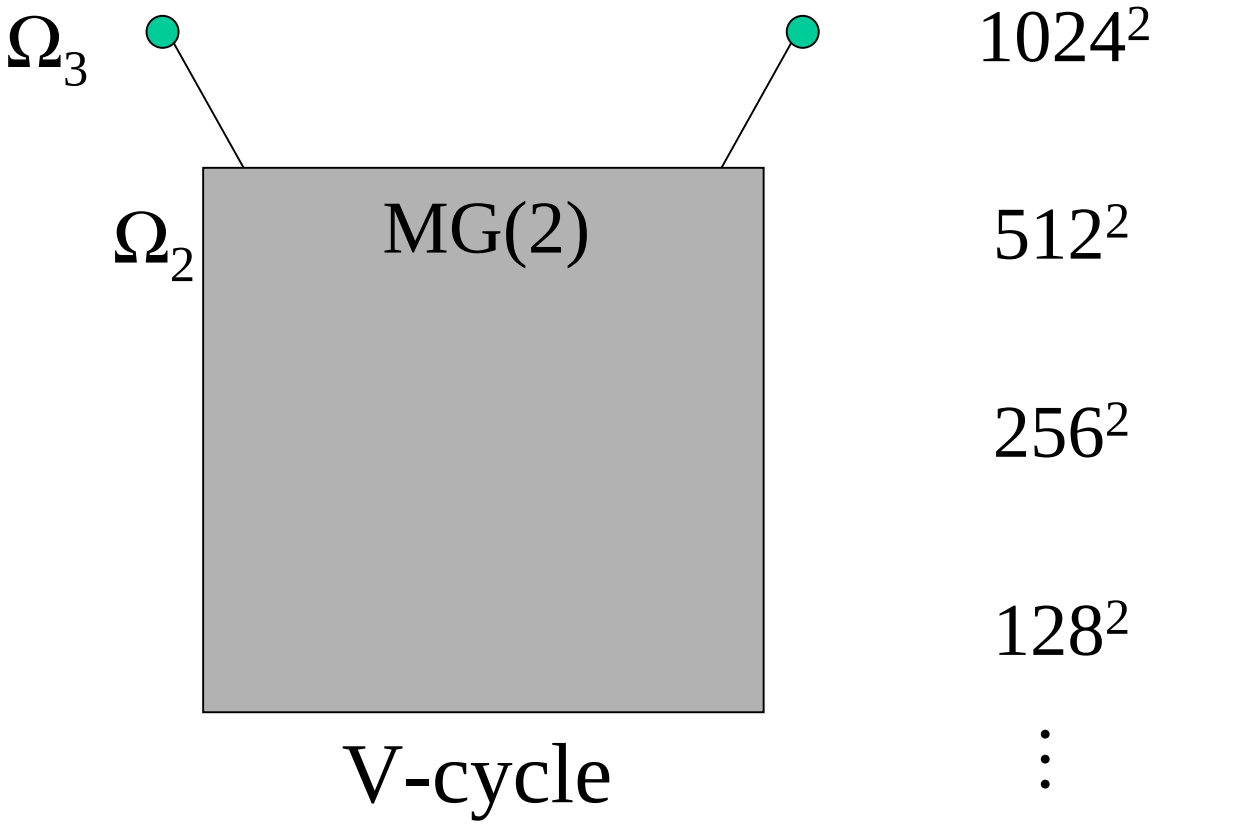
$$\mathbf{M}^{-1} = \mathbf{U}^T \mathbf{U}$$

Symmetric & Positive definite

Multigrid Method (1)

```
MG(IN k, INOUT x, IN b) {
    if (k is the coarsest) x = A_k^-1 b
    else {
        P_k x = Q_k x + b        // pre-smoothing
        b_0 = r(b - A_k x)       // coarse-grid
        x_0 = 0                  //     correction
        repeat (cycle) MG(k-1, x_0, b_0)
        x = x – p x_0
        P_k x = Q_k x + b        // post-smoothing
    }
}
```

Multigrid Method (2)



Parallelization using OpenMP

Naïve version

- Insert PARALLEL DO before each parallelizable DO loop.

Ex. Inner product

```
      IP = 0.0
!$OMP PARALLEL DO REDUCTION(+:IP)
      DO I = 1, N
         IP = IP + A(I) * B(I)
      END DO
```

OpenMP Optimization Techniques

- Merge several parallel regions
  - Reduce an overhead of creating and joining.
- Eliminate unnecessary barriers
  - `NOWAIT` clause
- Privatize variables
  - `PRIVATE`, `FIRSTPRIVATE` and `LASTPRIVATE`
- Determine a trade-off of parallel and sequential execution

Most of them can be detected by an OpenMP compiler

Example of Optimization (1)

```
      RR2 = 0
!$OMP PARALLEL PRIVATE(BETA)
!$OMP DO
      DO I = 1, N
          R1(I) = 0.0
      END DO
      CALL MG(...)                    // MG preconditioning
      CALL INNERPRO(N, R1, R, RR2)
      BETA = RR2 / RR1                // β
!$OMP DO
      DO I = 1, N
          P(I) = R1(I) + BETA * P(I)
      END DO                          // updating P
!$OMP END DO NOWAIT
!$OMP END PARALLEL
      RR1 = RR2
```

Example of Optimization (2)

```
      SUBROUTINE INNERPROD(N, A, B, IP)
!$OMP DO REDUCTION(+:IP)   // Orphaned directive
      DO I = 1, N
          IP = IP + A(I) * B(I)
      END DO
      RETURN
      END
```

Optimized version

Testing Platform

SGI Origin 2000

CPU : 195MHz R10000

nodes : 16

Cache : L1 32KB/32KB, L2 4MB

Memory : 4GB

OS : IRIX 6.5

- **Omni** OpenMP compiler pre1.2
- SGI **MIPSpro** compiler 7.3

Overhead of Creating a Parallel Region

Naïve

```
        DO J = 1, 100000  // Given at runtime
!$OMP PARALLEL DO
            DO I = 1, 8   // Given at runtime
                A(I) = A(I) + 2 * B(I)
            END DO
        END DO
```

Optimized

```
!$OMP PARALLEL
        DO J = 1, 100000
!$OMP DO
            DO I = 1, 8
                A(I) = A(I) + 2 * B(I)
            END DO
        END DO
!$OMP END PARALLEL
```

Overhead for Parallel Region

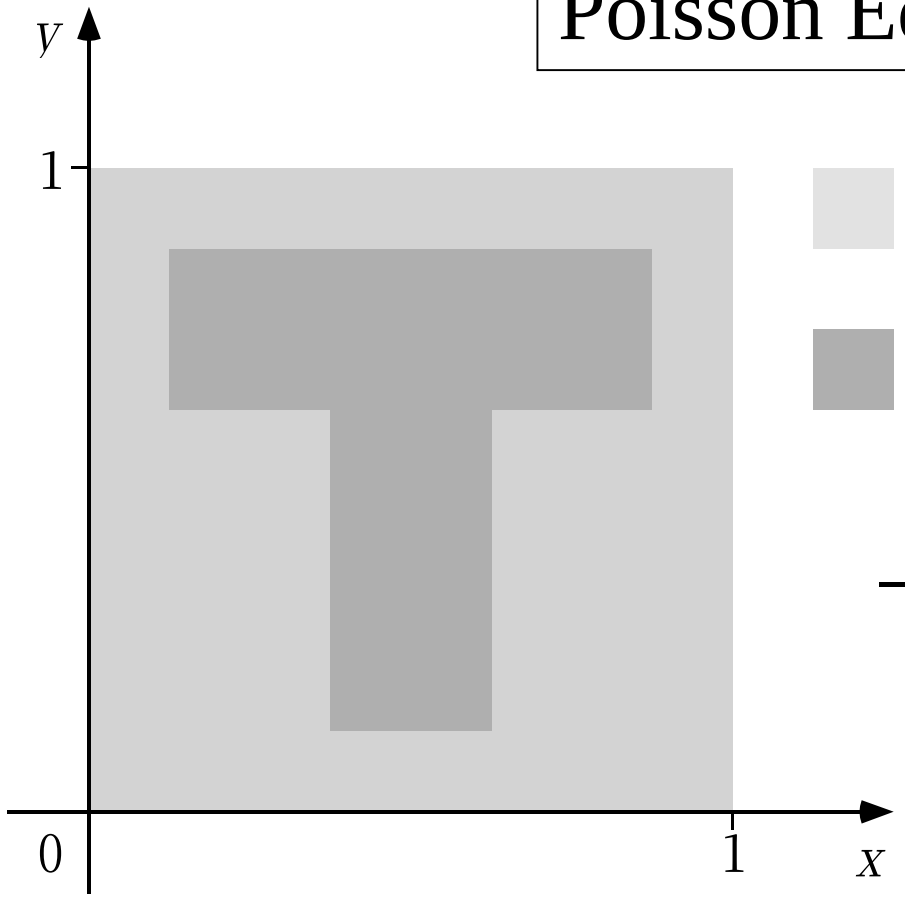
8 threads

	Naïve	Optimized
MIPSpro	2.90	2.07
Omni	46.76	2.86

[sec.]

Performance Evaluation of MGCG

Poisson Equation



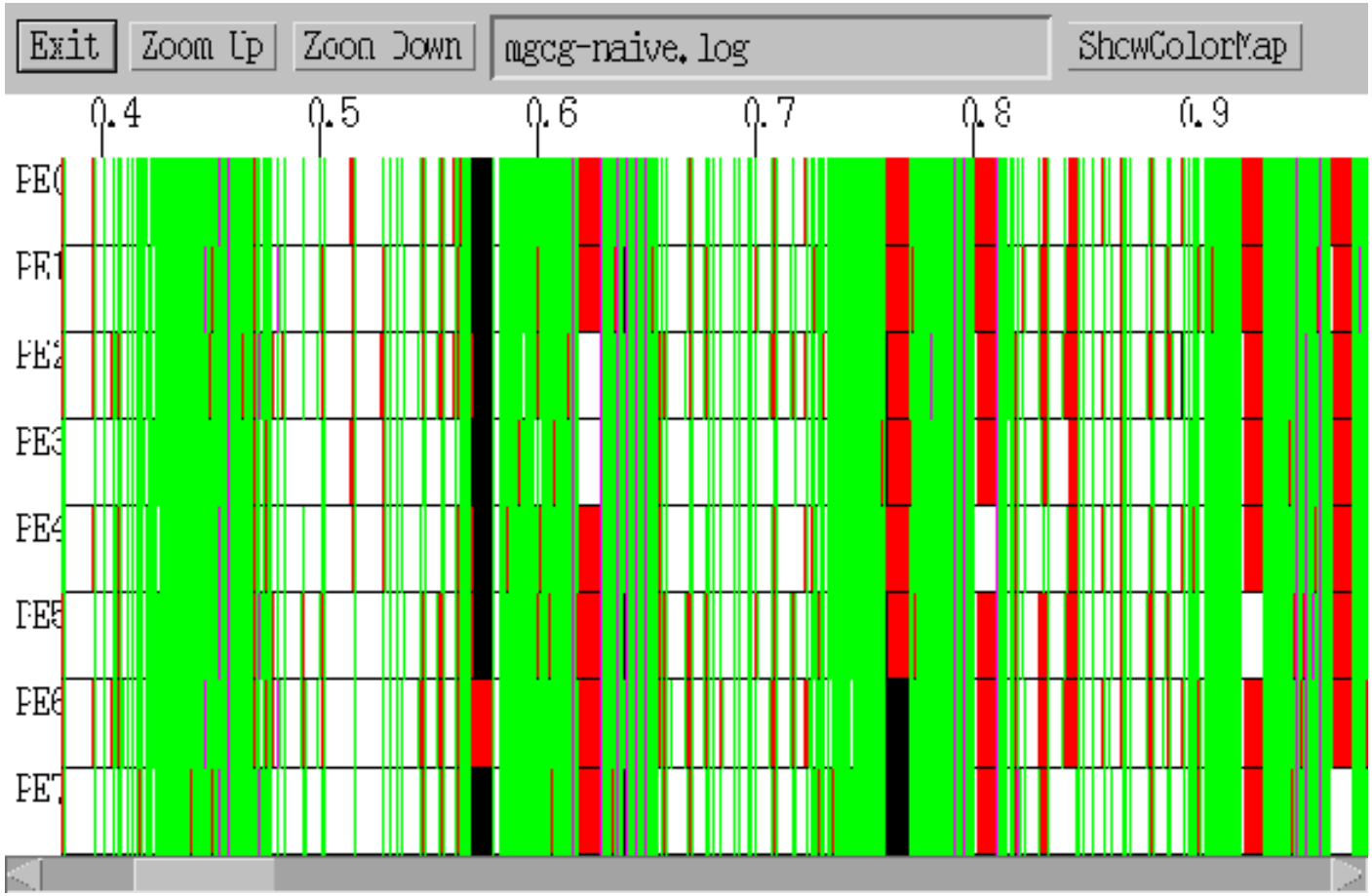
$$-\left(\frac{\partial}{\partial x}k\frac{\partial}{\partial x}+\frac{\partial}{\partial y}k\frac{\partial}{\partial y}\right)u=f$$

$$u=0 \quad \text{on} \quad \Gamma_1$$

Profiling

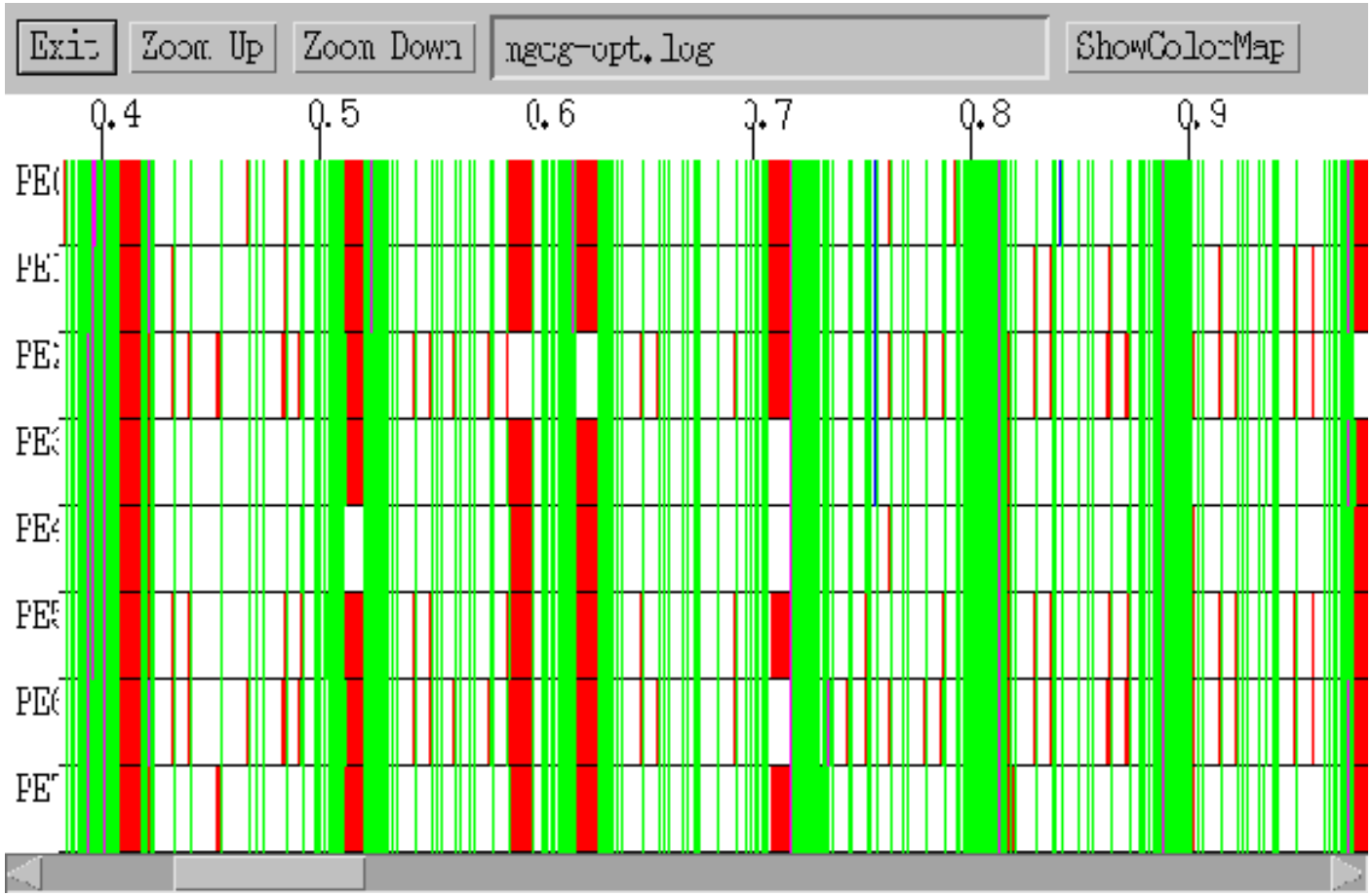
- Omni OpenMP compiler provides a very **light-weight** profiling option.
- Each event related to OpenMP consists of only **two double words**.
- **`tlogview`** displays the profiled log, which is included by the Omni compiler distribution.

Profiling – Naive



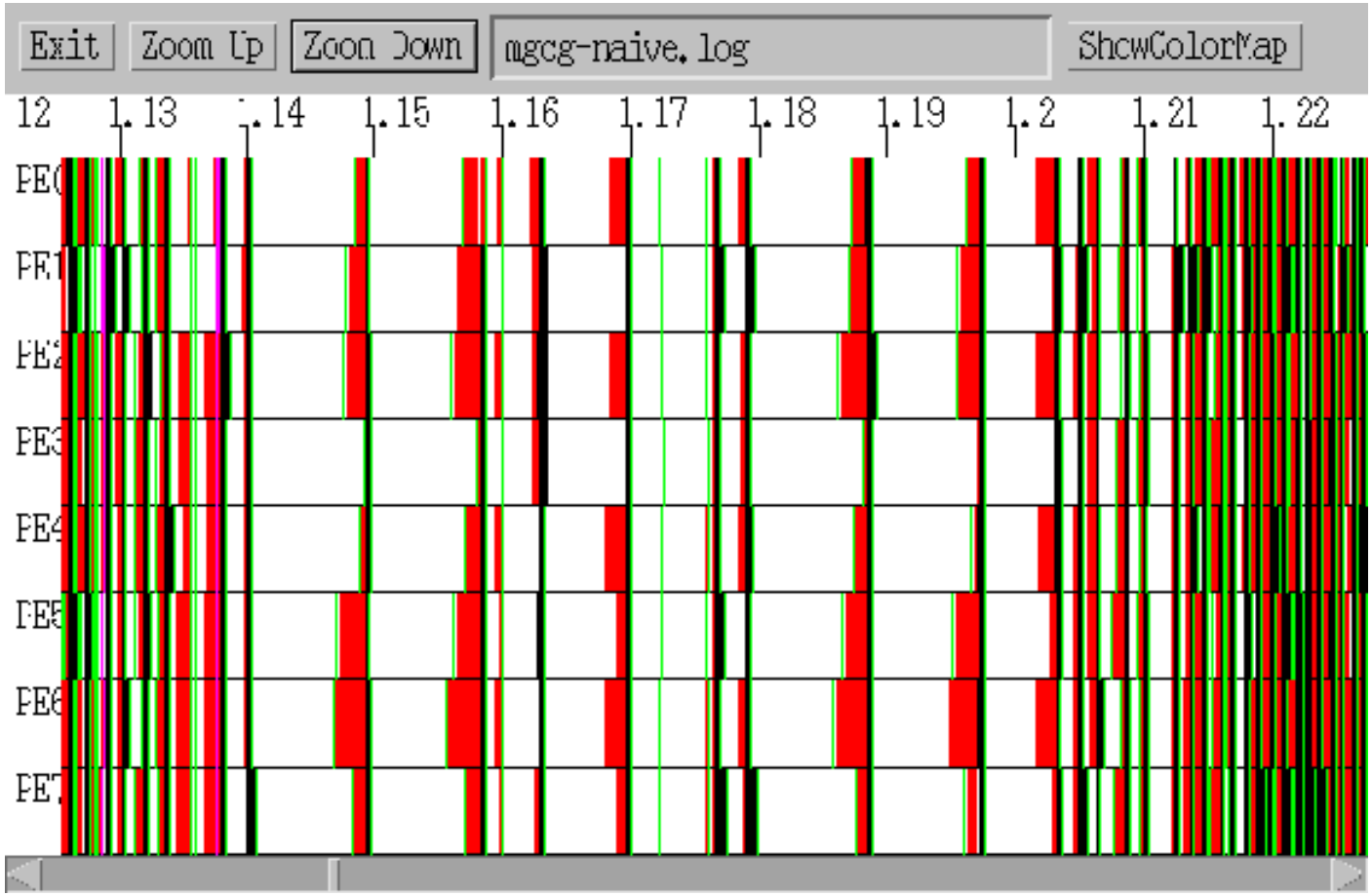
Parallel execution | Barrier | Loop initialization | Serial part

Profiling – Optimized



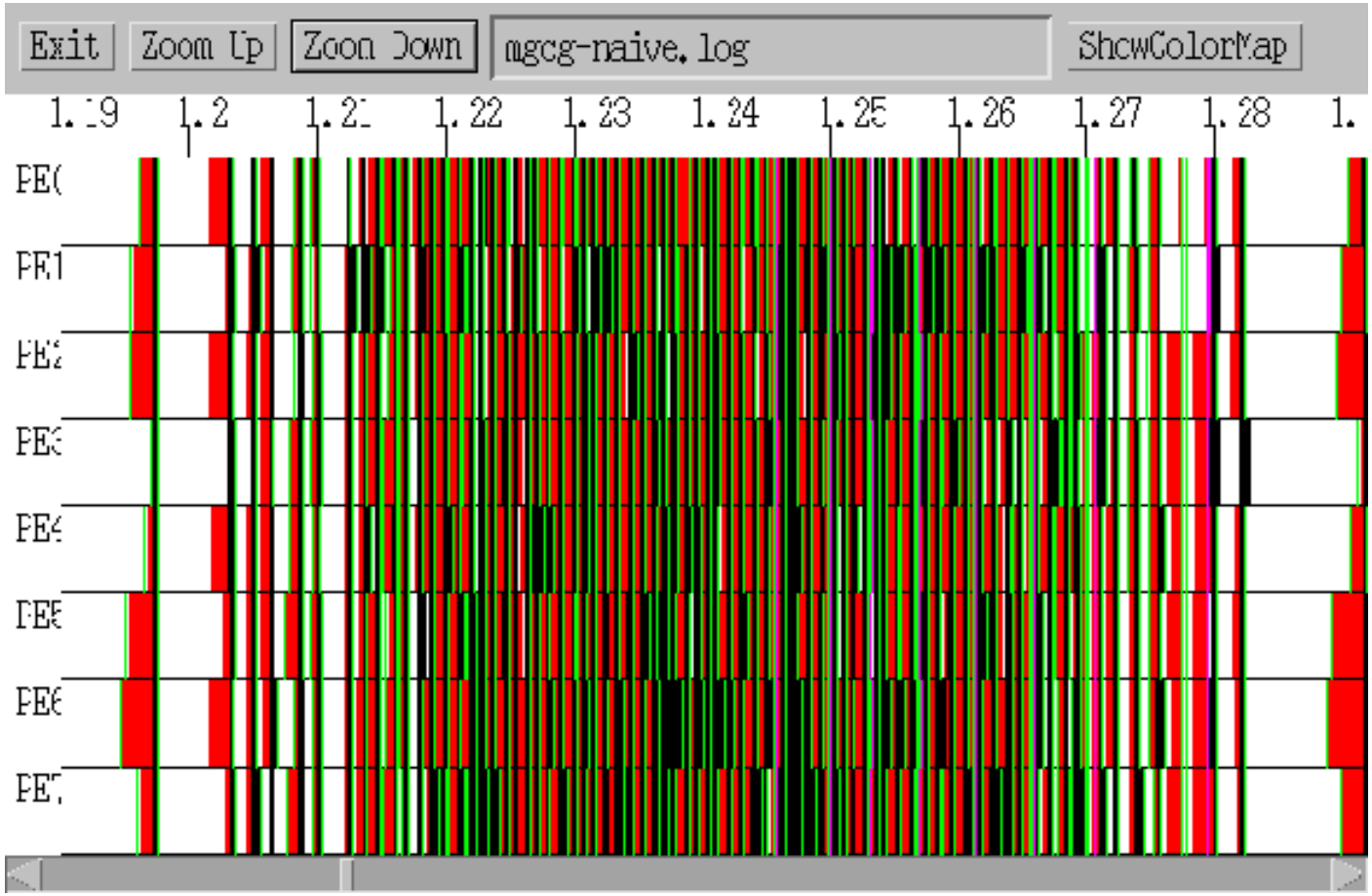
Parallel execution | Barrier | Loop initialization | Serial part

Profiling – Naive (Magnified;CG)



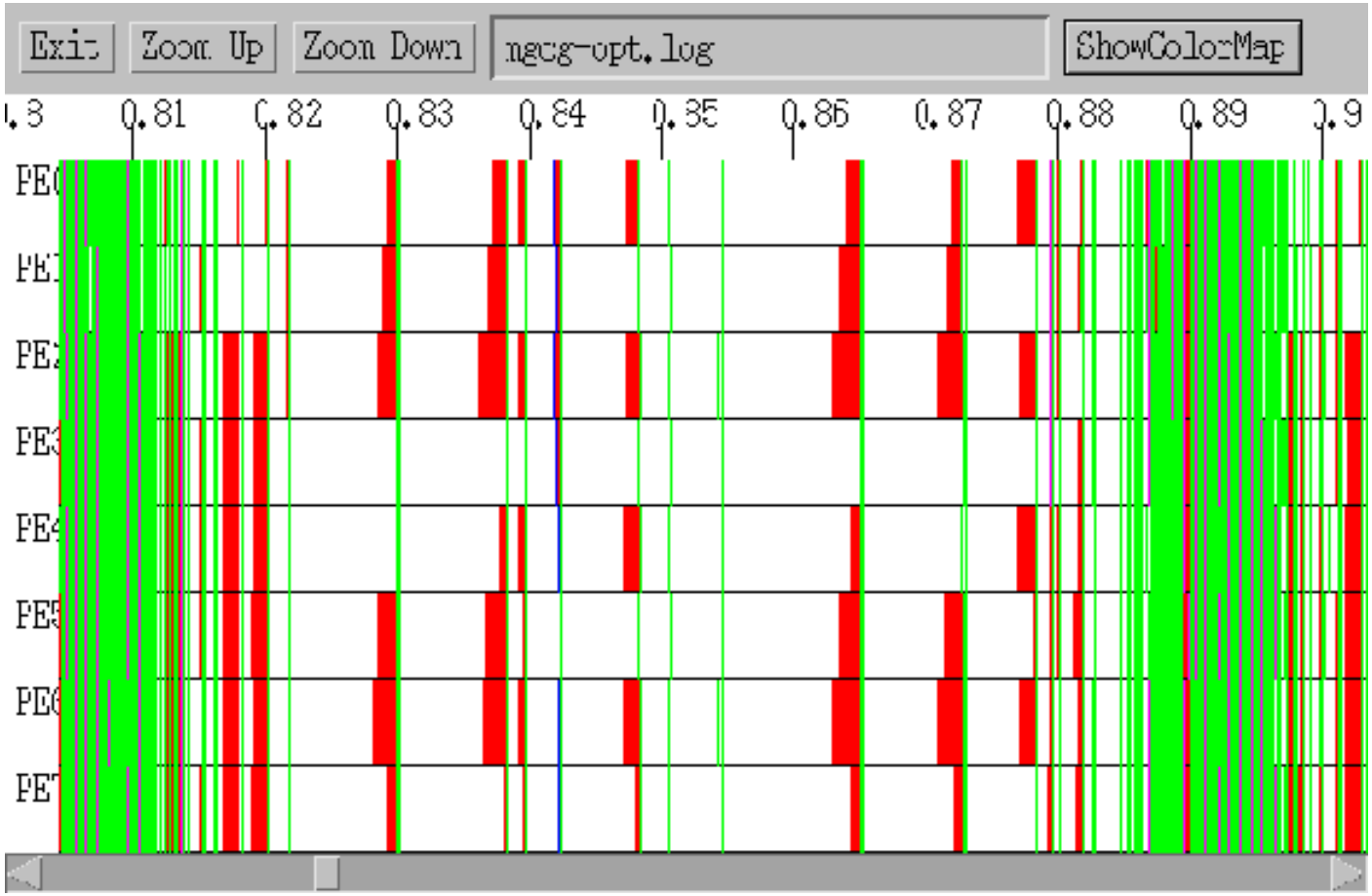
Parallel execution | Barrier | Loop initialization | Serial part

Profiling – Naive (Magnified;MG)



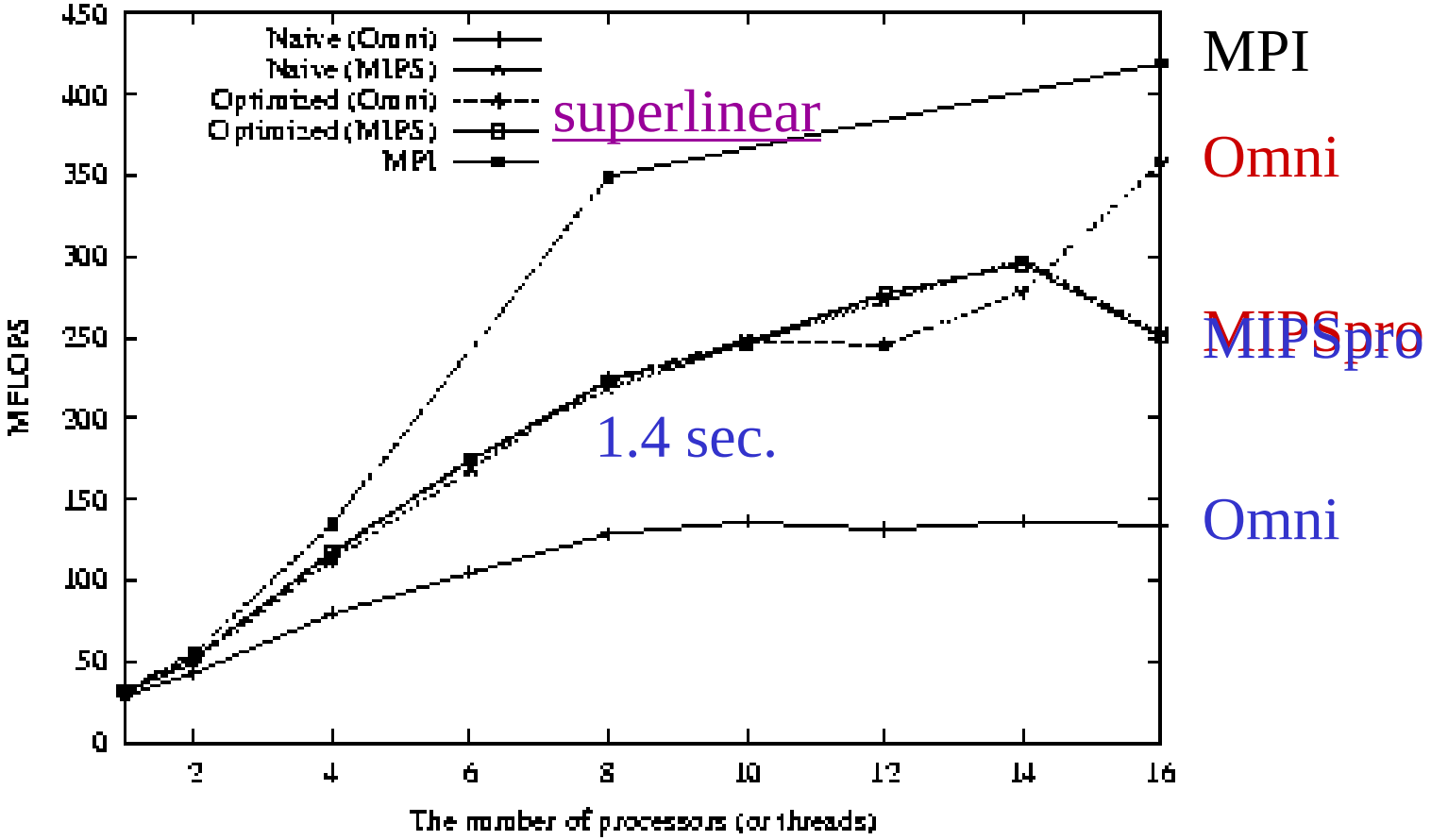
Parallel execution | Barrier | Loop initialization | Serial part

Profiling – Optimized (Magnified)



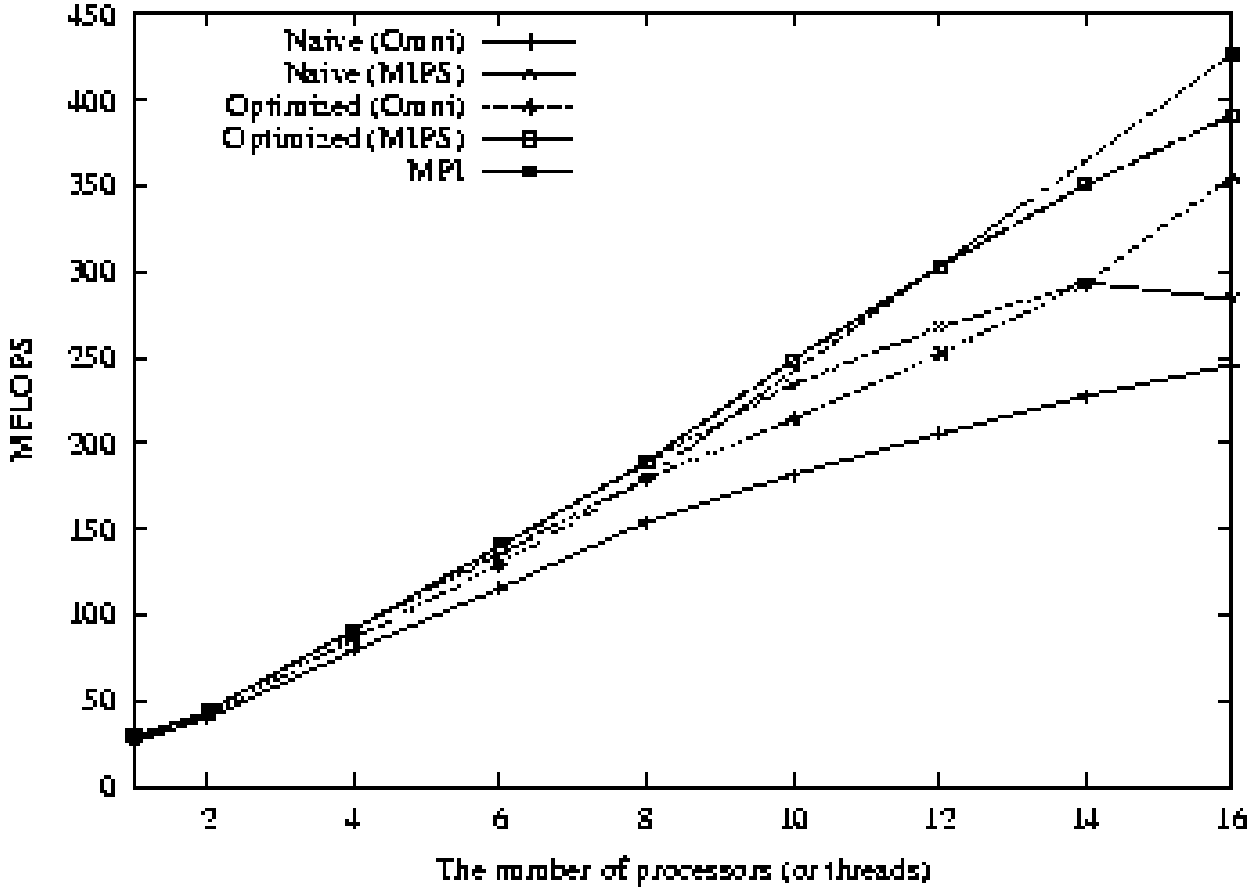
Parallel execution | Barrier | Loop initialization | Serial part

Floating-point Performance (512x512)



MPI can utilize cache memory efficiently.

Floating-point Performance (1024x1024)



MPI
MIPSpro
Omni

MIPSpro
Omni

MPI & MIPSpro scale up to 16 procs.

Proposals for OpenMP Specification

- `READONLY` clause
  - Variables with `FIRSTPRIVATE` will be undefined after the parallel region.
  - This clause ensures to keep the original value.
- Necessity to specify the order of `REDUCTION`
  - # iterations until convergence may differ every time.
  - Reproducibility is quite important for testing and debugging.

Possible Problems of Optimized Version

- Local variables have SAVE attribute by default using (conventional) Fortran compiler. (Ex. HITACHI)
- This causes a problem to call a subroutine within a parallel region.
  - Unexpected data race may occur.

Summary (1)

- Optimization impact depended on compilers.
  - Omni showed almost double the performance.
  - MIPSpro showed little difference of performance, though the optimized one scales well.
- How to exploit the locality?
  - The program written in MPI exploits locality of memory explicitly and efficiently.

Summary (2)

- MGCG method is important for also benchmarking including various loop length and reduction.
- This benchmark program will be included by the Omni compiler distribution.

Omni OpenMP Compiler 1.2s

- Now Omni OpenMP compiler 1.2s is available

  http://pdplab.trc.rwcp.or.jp/Omni/
- Solaris, Linux, IRIX, AIX, FreeBSD, Windows NT/2000
- Pthreads, Solaris Threads, Sproc, StackThreads for nested parallelism
- Cluster-enabled OpenMP using SCore/SCASH
- CD-ROM is available at RWCP booth in SC2000.