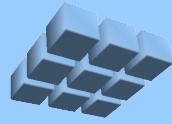
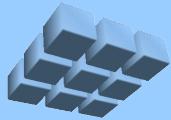


# **The Cmpware CMP-DK 3.0: Cell BE Version**

Cmpware, Inc.

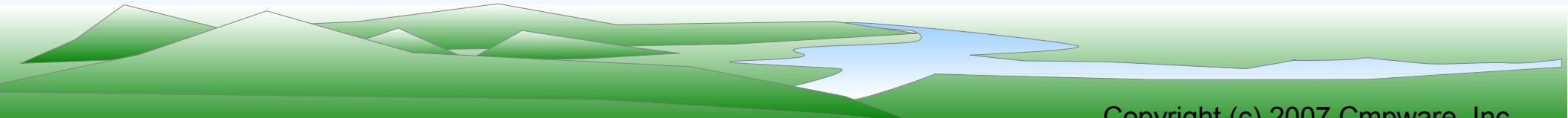
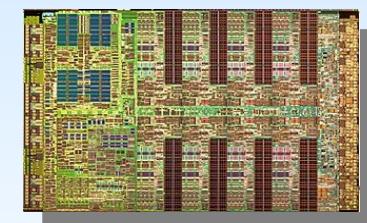


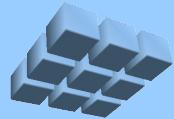
- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis
- IX. Overview



# Introduction

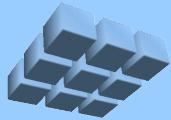
- *Cmpware CMP-DK* for the *Cell BE*
  - Uses *Cmpware's* powerful multicore modeling technology
  - A simulation based software development environment for the Cell BE
  - Special Cell BE displays to aid in code partitioning and optimization of
  - Complements existing tools





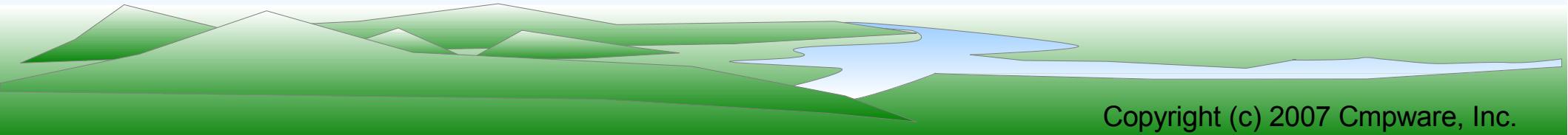
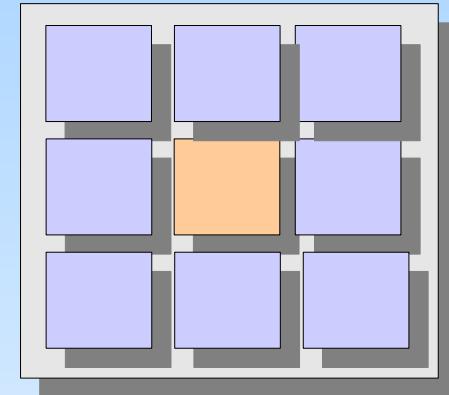
# The Cmpware Approach

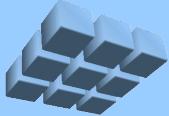
- Multicore simulation
  - Software development on simulator
  - High level of execution control
  - Run-time execution statistics
  - Simpler (and often cheaper) than HW
- Powerful multicore IDE
  - Easy access to all system data
  - Specifically designed for multicore



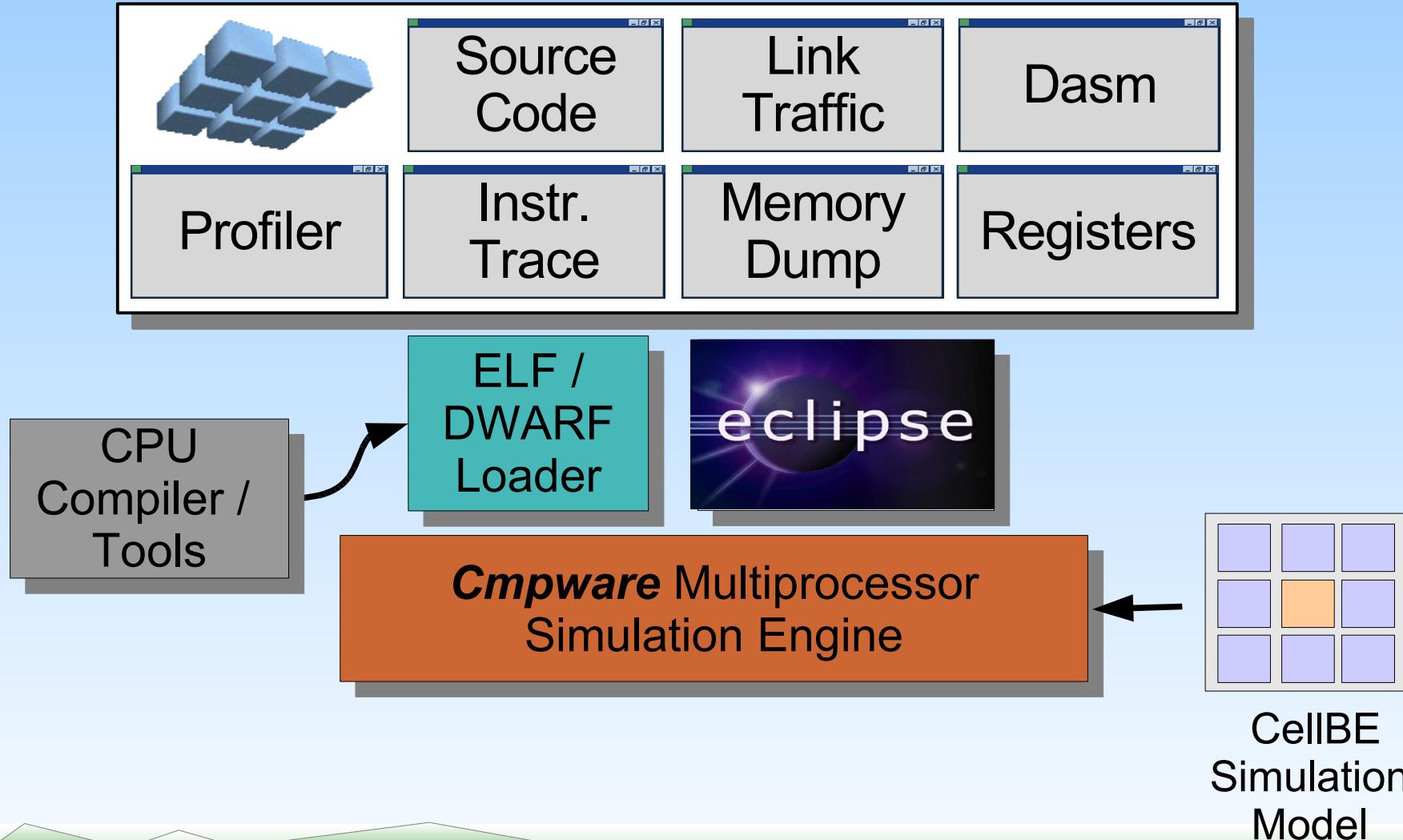
# The Cell BE Simulation Model

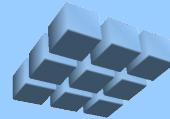
- Cmpware *Cell BE* simulation model:
  - 64-bit *PowerPC* core
  - *PowerPC FP* extensions
  - Eight *SPU* cores
  - System level model ('glue')
- Supplies all IDE display data
- Built-in assemblers and disassemblers
- **4M+** operations per second

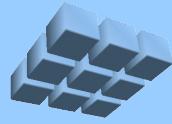




# *Cmpware* CMP-DK IDE

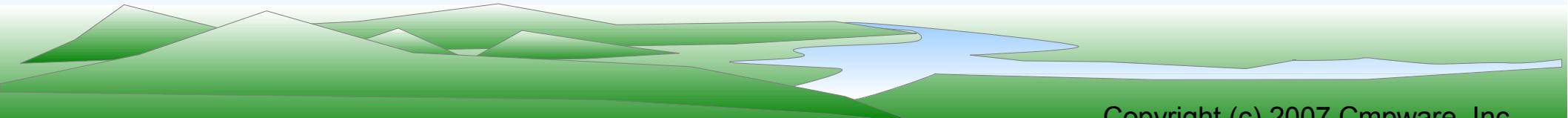


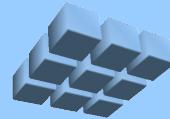
- 
- I. Introduction
  - II. Installing the *Cmpware CMP-DK*
  - III. Application I: A Simple Program
  - IV. Application II: Shared Memory
  - V. Application III: Mailboxes
  - VI. Optimization I: SPU Dependencies
  - VII. Optimization II: SPU Dual Issue
  - VIII. System Level Analysis
  - IX. Overview



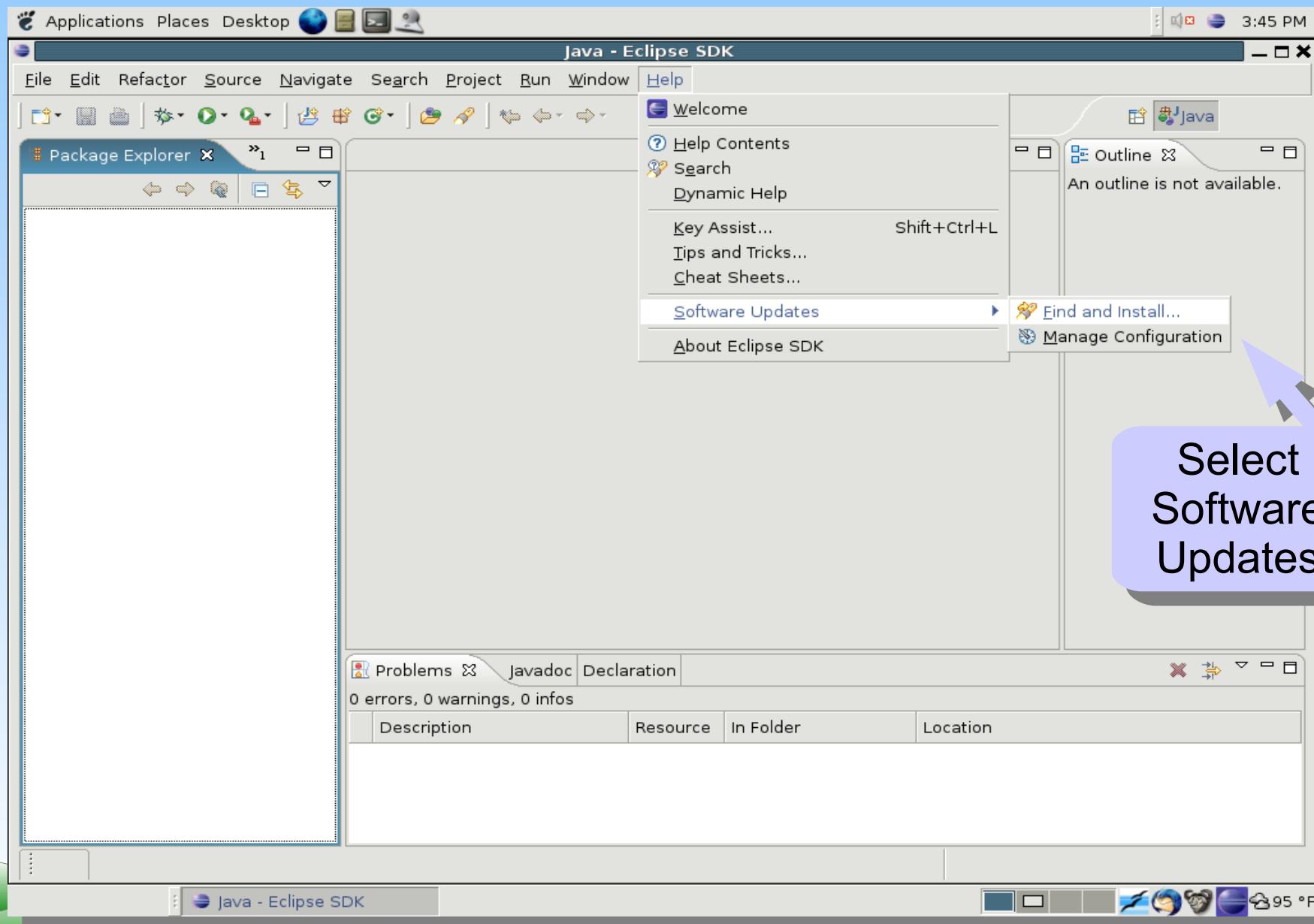
# Installing the Cmpware CMP-DK

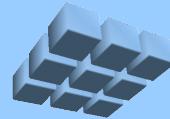
- An *Eclipse 'plugin'*
  - Approximately 1.5 MB (total)
  - Installs from an *Eclipse 'Update Site'*:  
*<http://www.cmpware.com/cellbe/>*
  - Up and running in seconds\*
  - Works with other existing *Eclipse* installations
  - 'C' Development Toolkit (CDT) recommended
- \* requires license code from *Cmpware*



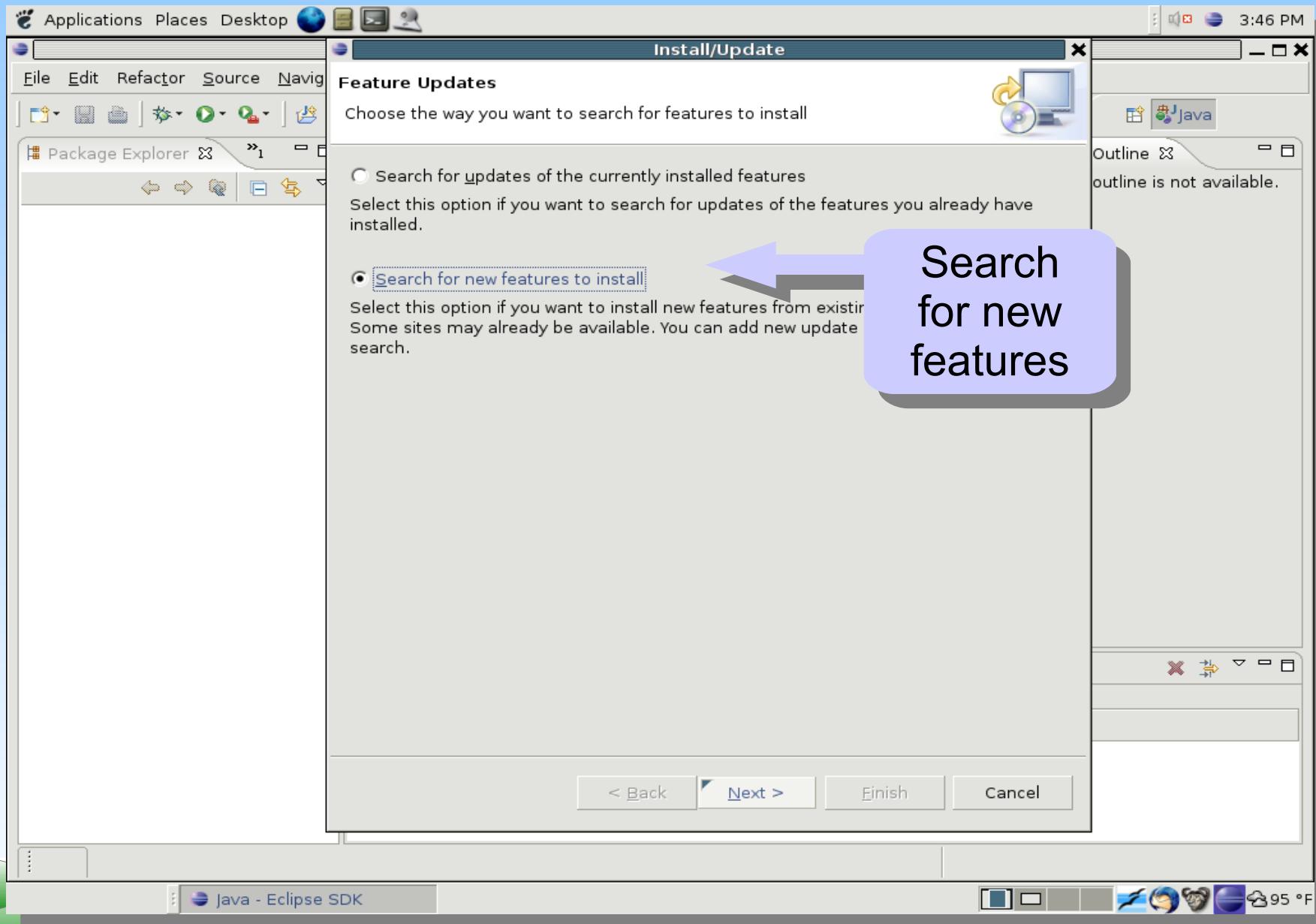


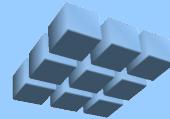
# Installation: Step 1



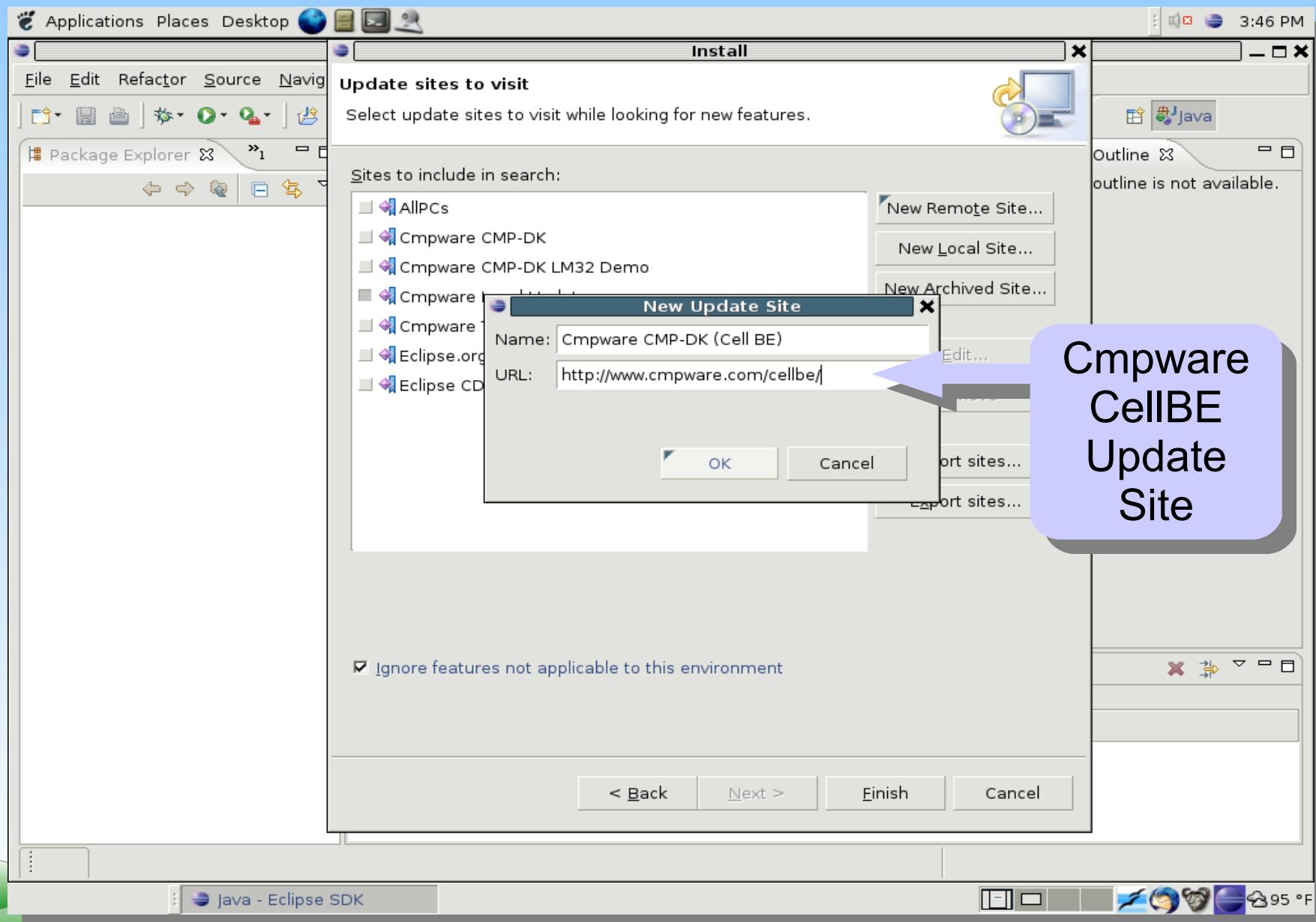


# Installation: Step 2

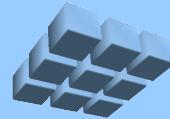




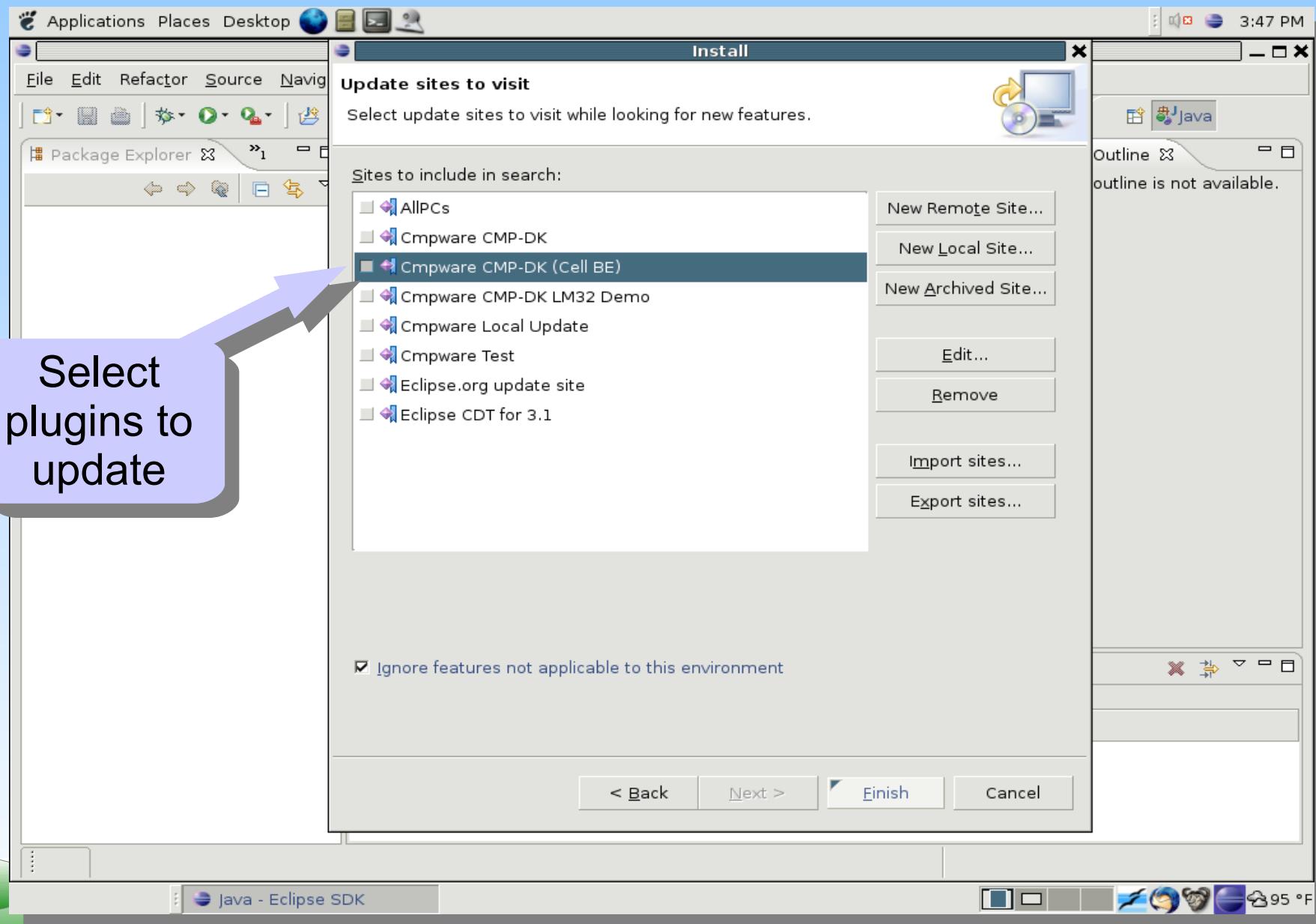
# Installation: Step 3

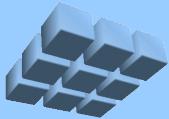


Cmpware  
CellBE  
Update  
Site

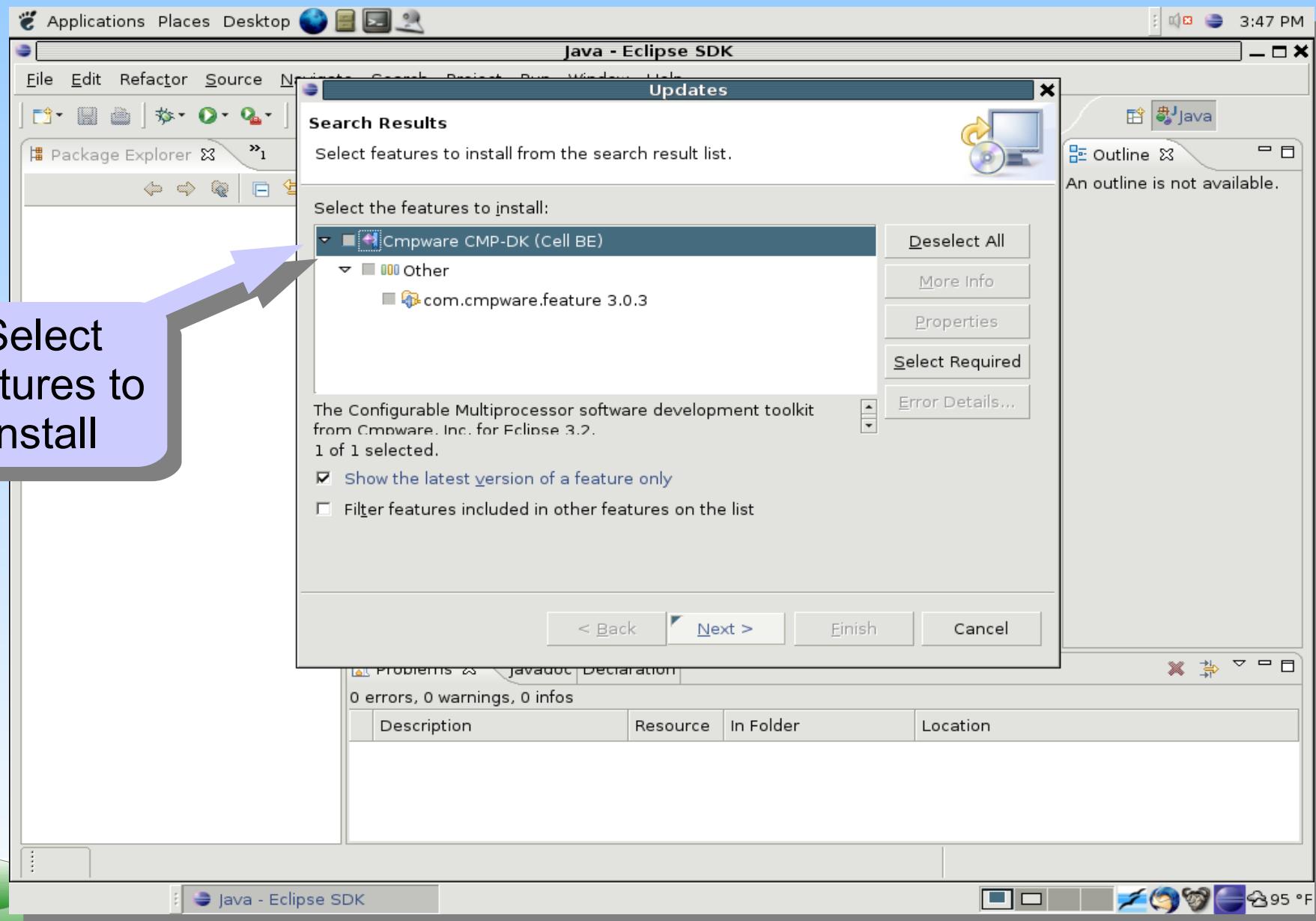


# Installation: Step 4

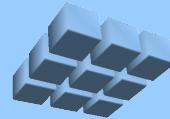




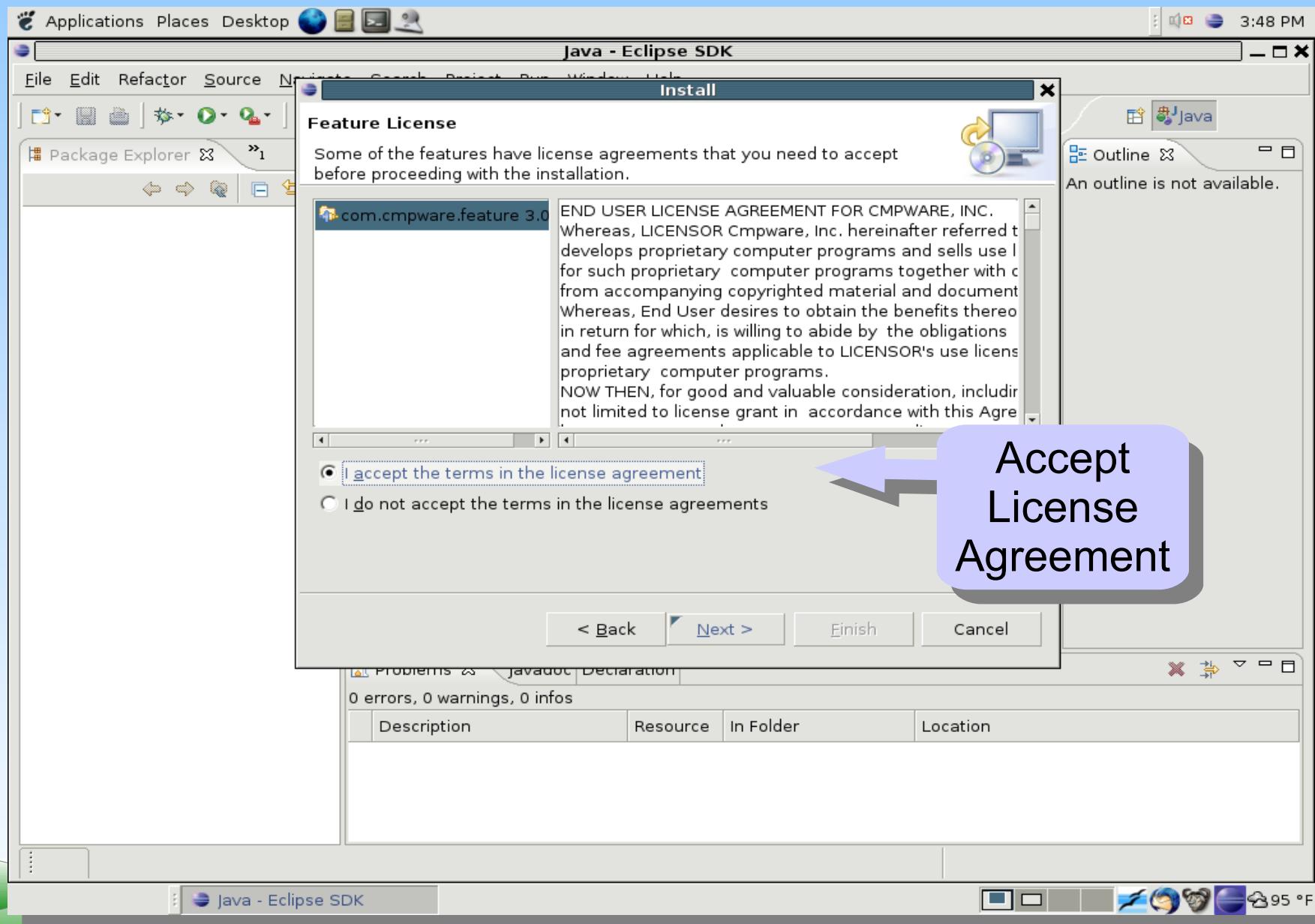
# Installation: Step 5

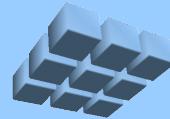


Select  
features to  
install

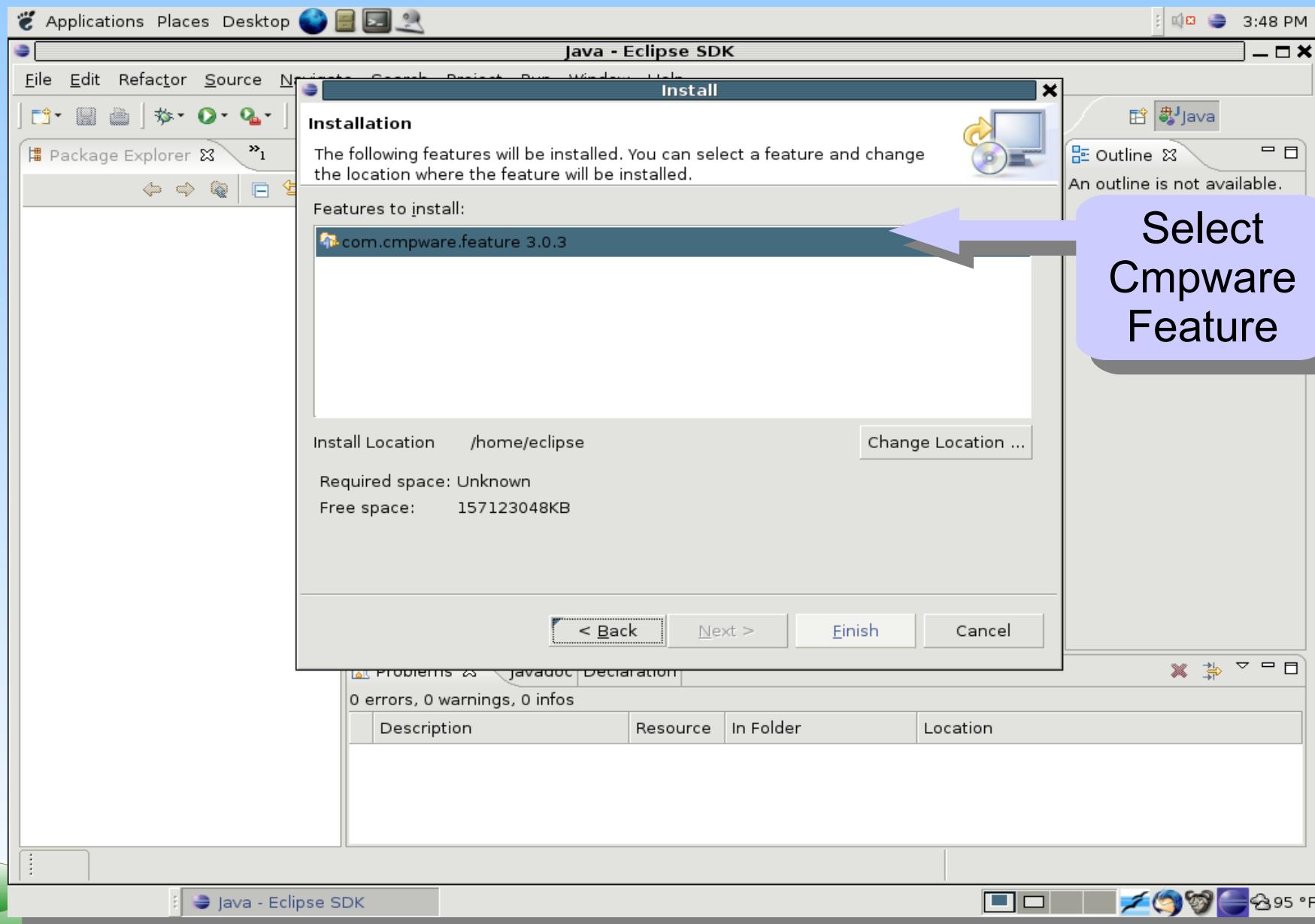


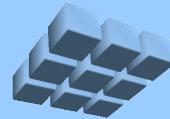
# Installation: Step 6



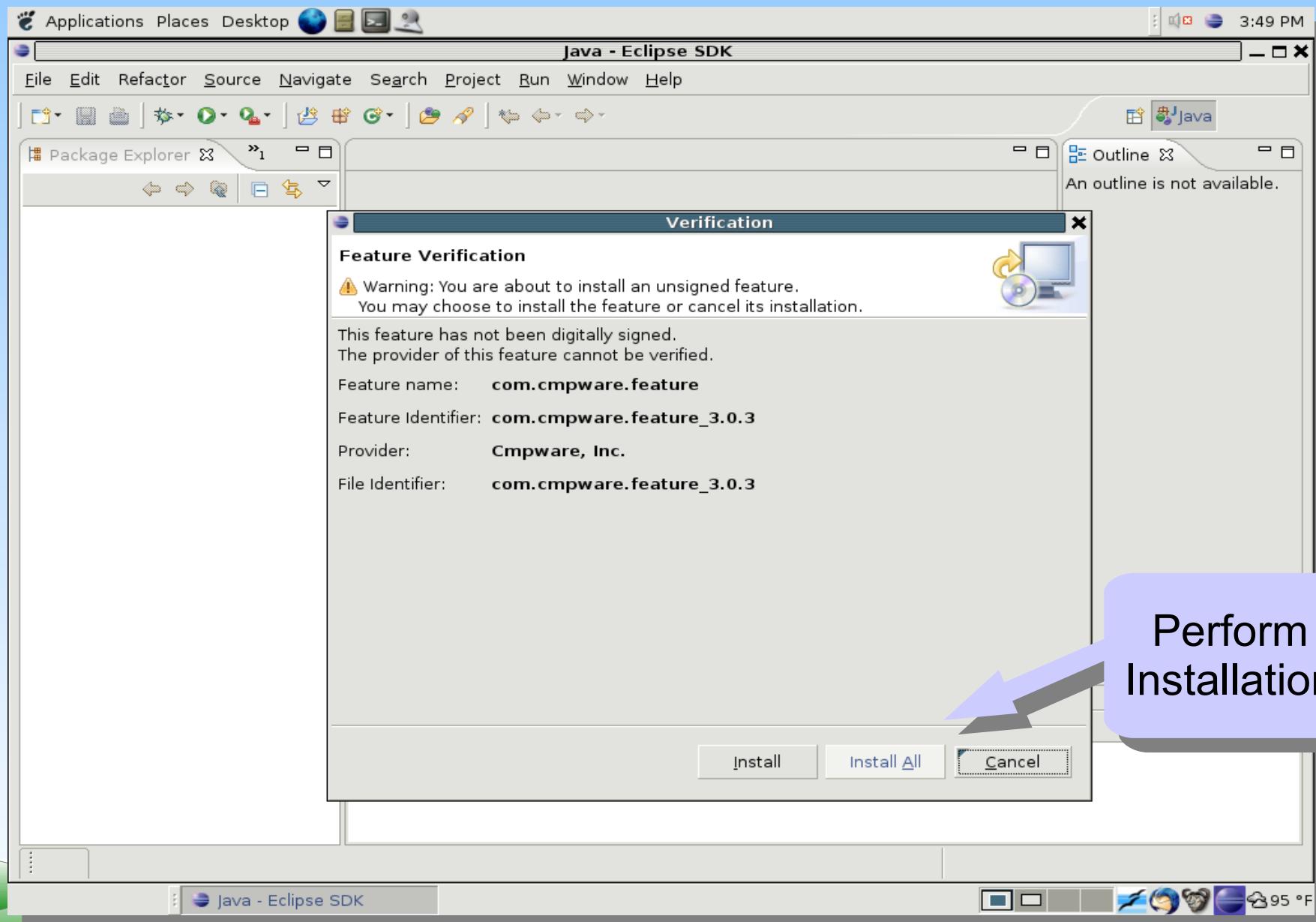


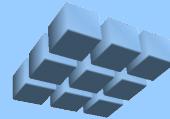
# Installation: Step 7



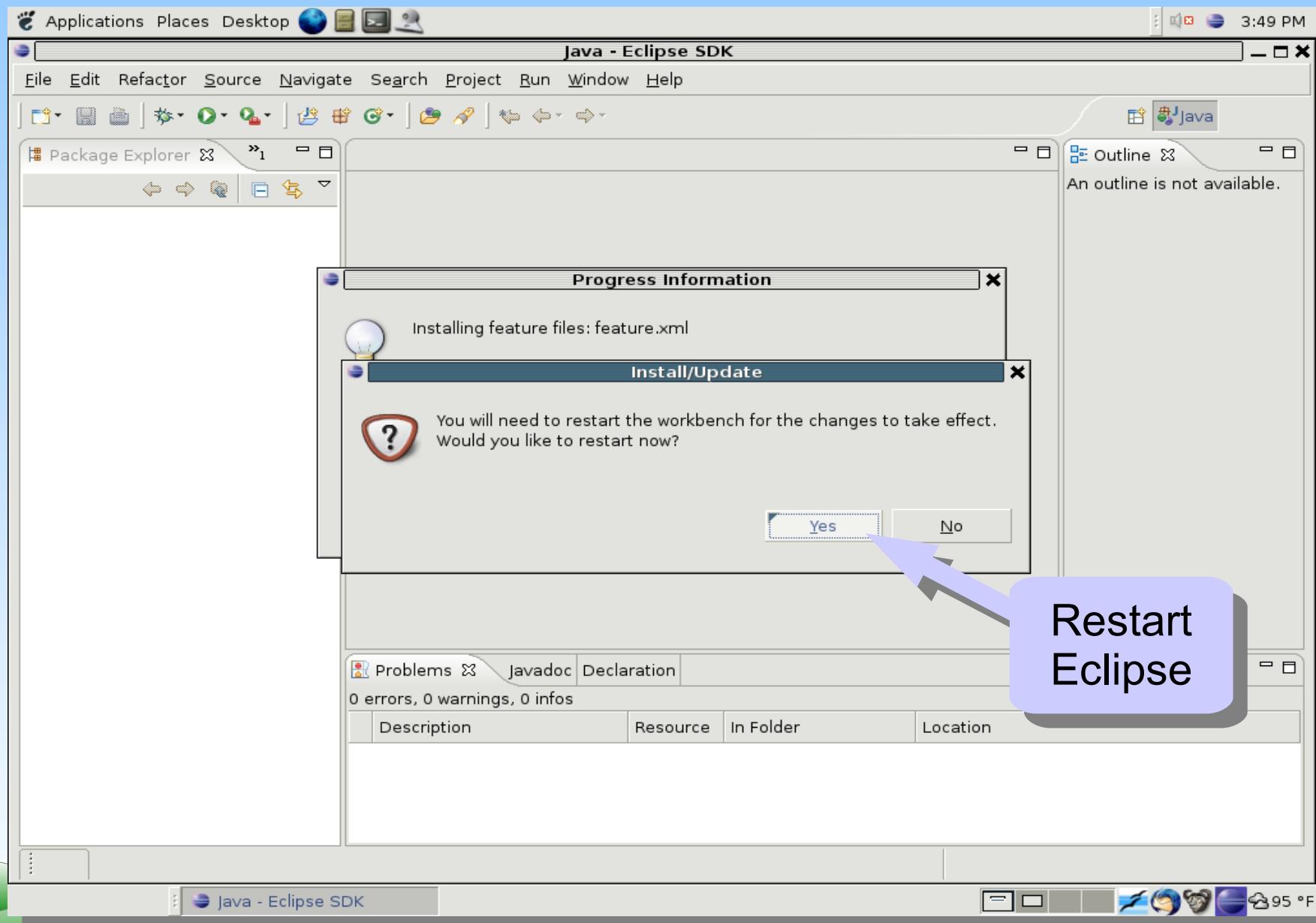


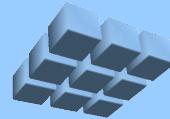
# Installation: Step 8



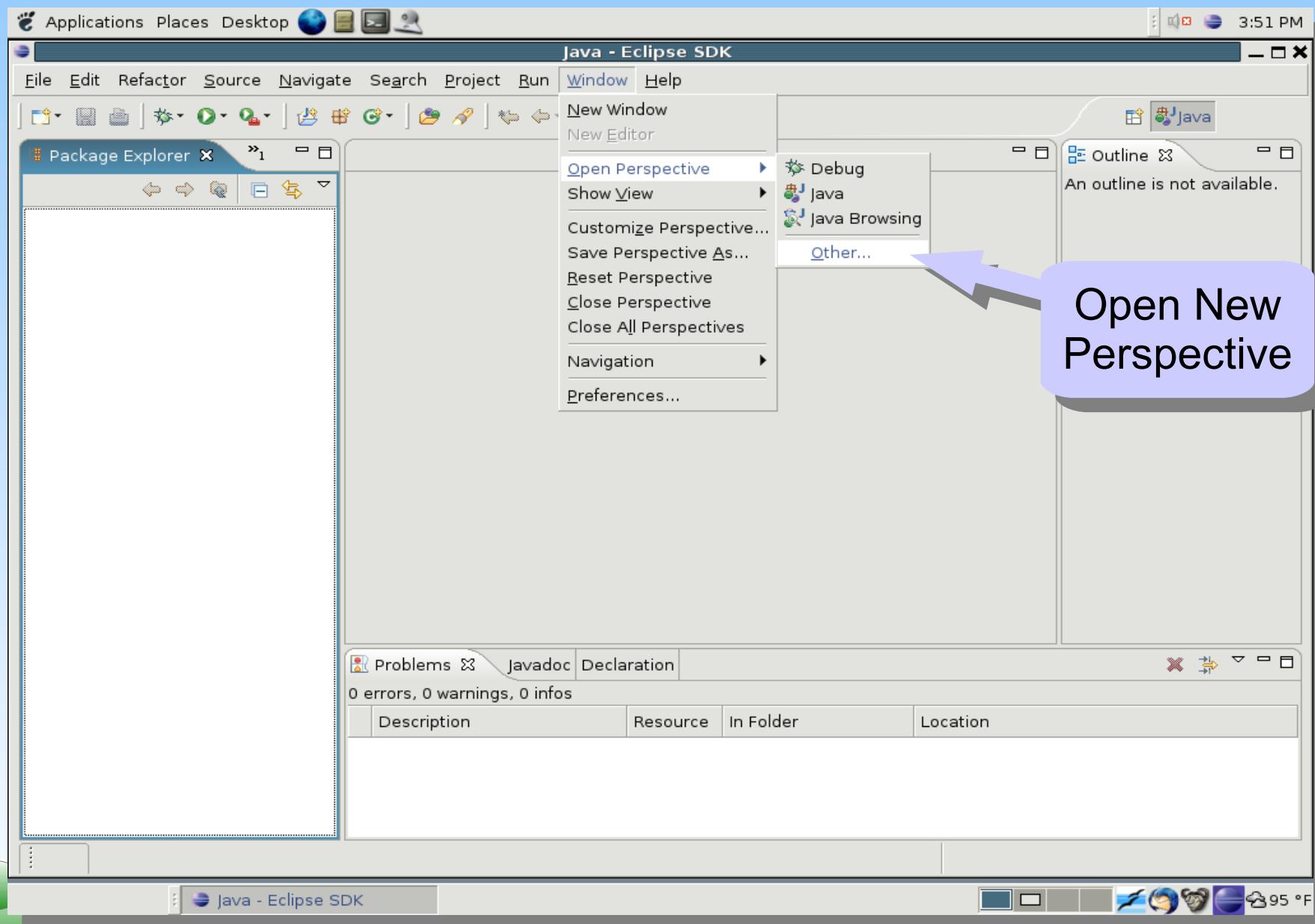


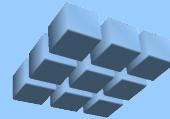
# Installation: Step 9



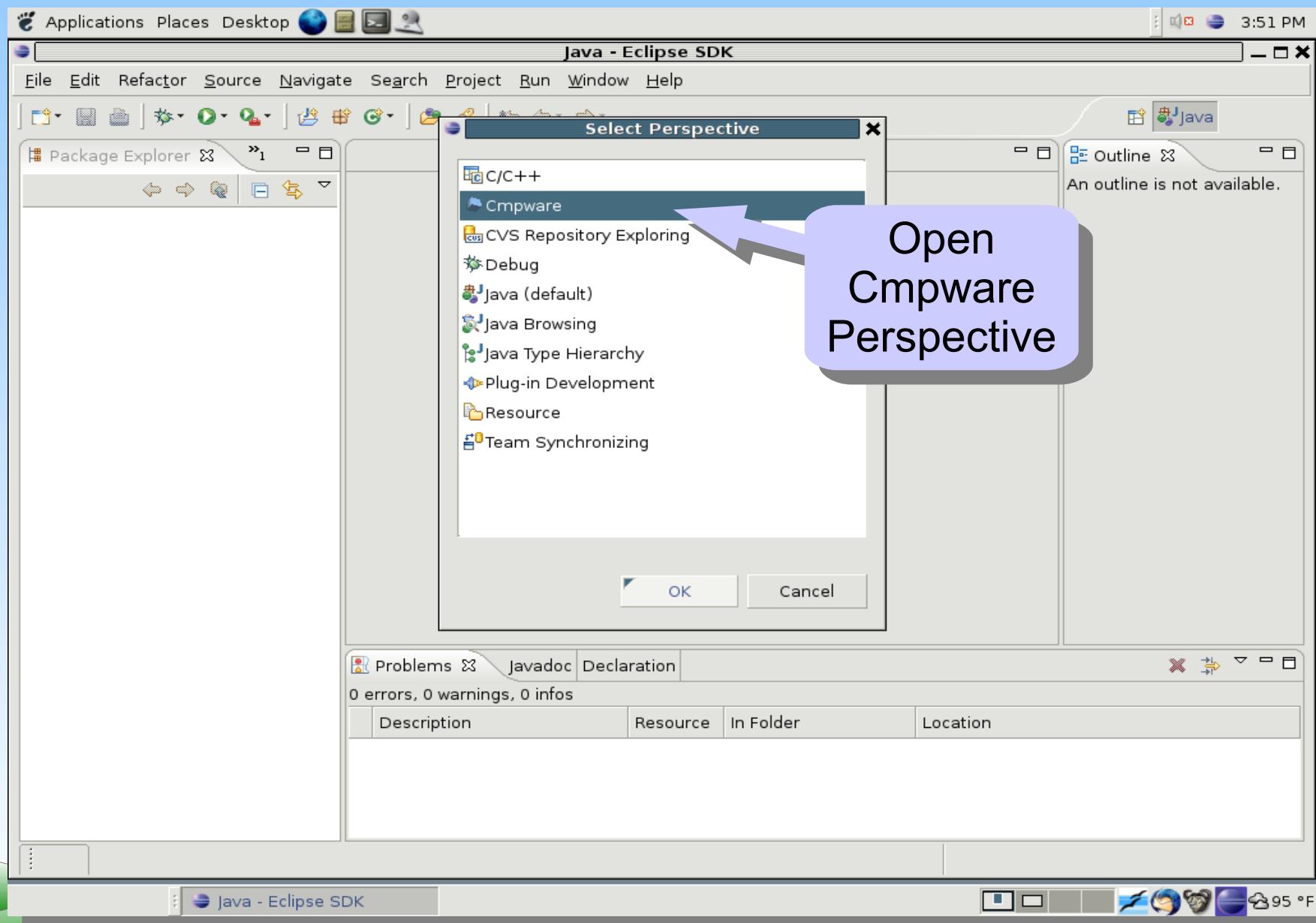


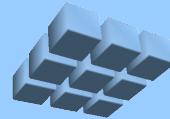
# Installation: Step 10



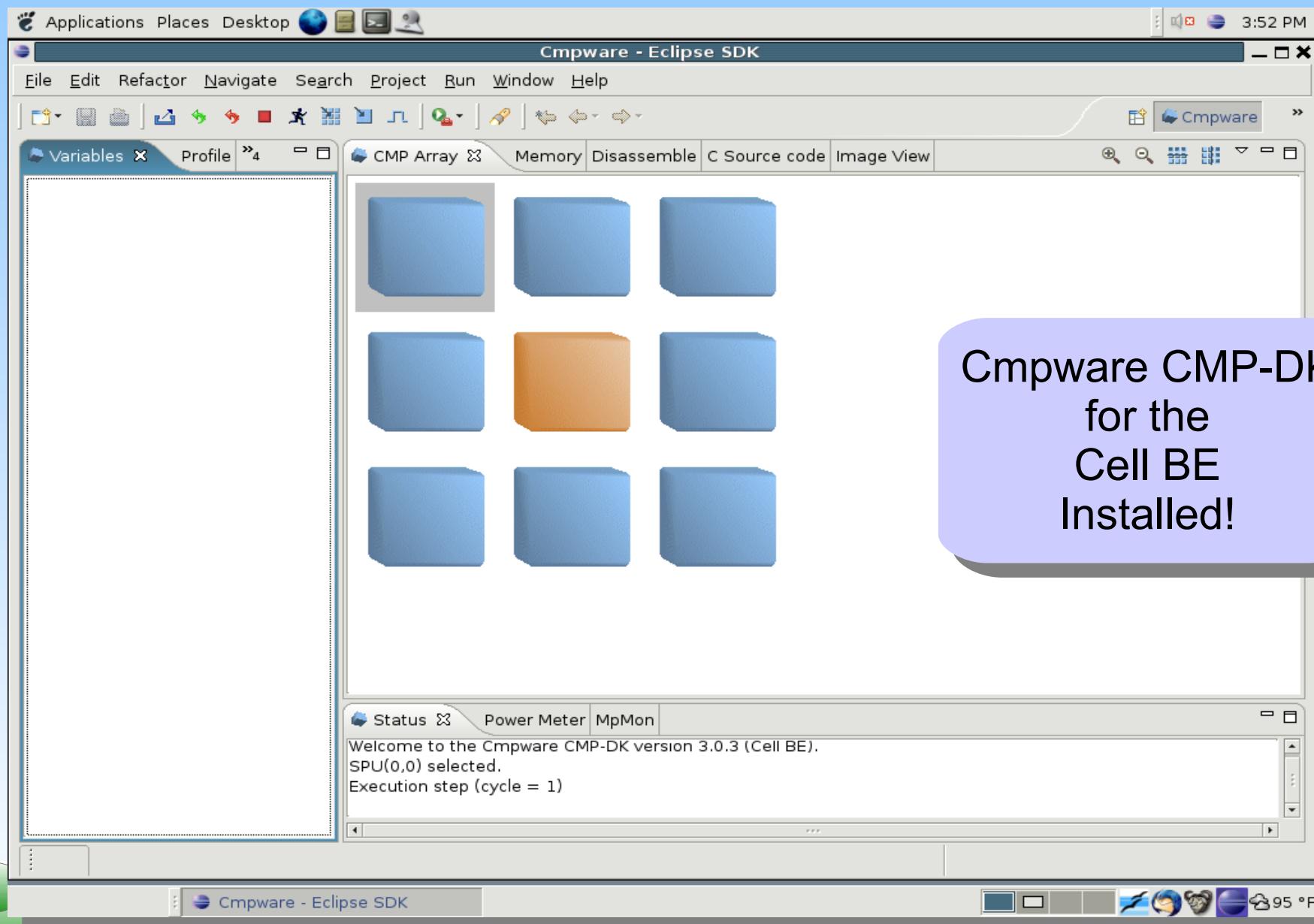


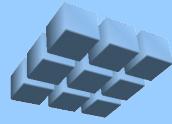
# Installation: Step 11



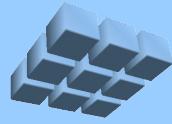


# Installation: Cmpware CMP-DK





- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis
- IX. Overview



# Application I: A Simple Program

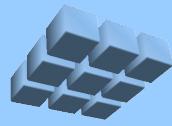
- Compiled for both PPU and SPUs
- Tests compiler and *Cmpware* installation
- Uses IBM Cell tools for Windows / Cygwin

(<http://cellbe-cygwin.cvs.sourceforge.net/cellbe-cygwin/cellbe-cygwin/>)

```
/*
**  A simple test program
**
** Copyright (c) 2007 Cmpware, Inc. All Rights Reserved.
*/
int main(int argc, char *argv[]) {
    int i;
    float a = 1.0;

    for(i=0; i<1000; i++)
        a = a + 1.0;

} /* end main() */
```



# Application I: CDT Makefile

The screenshot shows the Eclipse CDT (C/C++) Perspective interface. The central view displays a Makefile with the following content:

```
## This makefile produces a binary executable (ELF) file
## for a PPE (PowerPC) executable and raw binary executables
## for the SPEs in the Cell BE.
##
## Copyright (c) 2007 Cmpware, Inc. All rights reserved.
##

CLASSPATH = C:/Users/Cmpware/Devel/com.cmpware.ide/bin
TOOLPATH = C:/cygwin/opt/cell/bin
SPU = com.cmpware.cmp.models.SPU
CFLAGS = -g -c
LDFLAGS = -g -e 0x0000 -T Cmpware.lnk
PPE32 = -m32

all: simple mailbox mandelbrot mandelbrot_mb

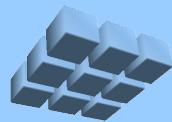
simple:
    ${TOOLPATH}/spu-gcc ${CFLAGS} -o bin/Simple_spu.o Simple.c
    ${TOOLPATH}/spu-ld ${LDFLAGS} -o Simple_spu.elf bin/Simple_spu
    ${TOOLPATH}/spu-objdump -xdw Simple_spu.elf > bin/Simple_spud
    ${TOOLPATH}/ppu-gcc ${CFLAGS} -o bin/Simple_ppu.o Simple.c
    ${TOOLPATH}/ppu-ld ${LDFLAGS} -o Simple_ppu.elf bin/Simple_ppu
    ${TOOLPATH}/ppu-objdump -xdw Simple_ppu.elf > bin/Simple_ppud

mailbox:
    ${TOOLPATH}/ppu-gcc ${CFLAGS} -o bin/Mailbox_ppu.o Mailbox_ppu
    ${TOOLPATH}/ppu-ld ${LDFLAGS} -o Mailbox_ppu.elf bin/Mailbox_ppu
    ${TOOLPATH}/ppu-objdump -xdw Mailbox_ppu.elf > bin/Mailbox_ppud
```

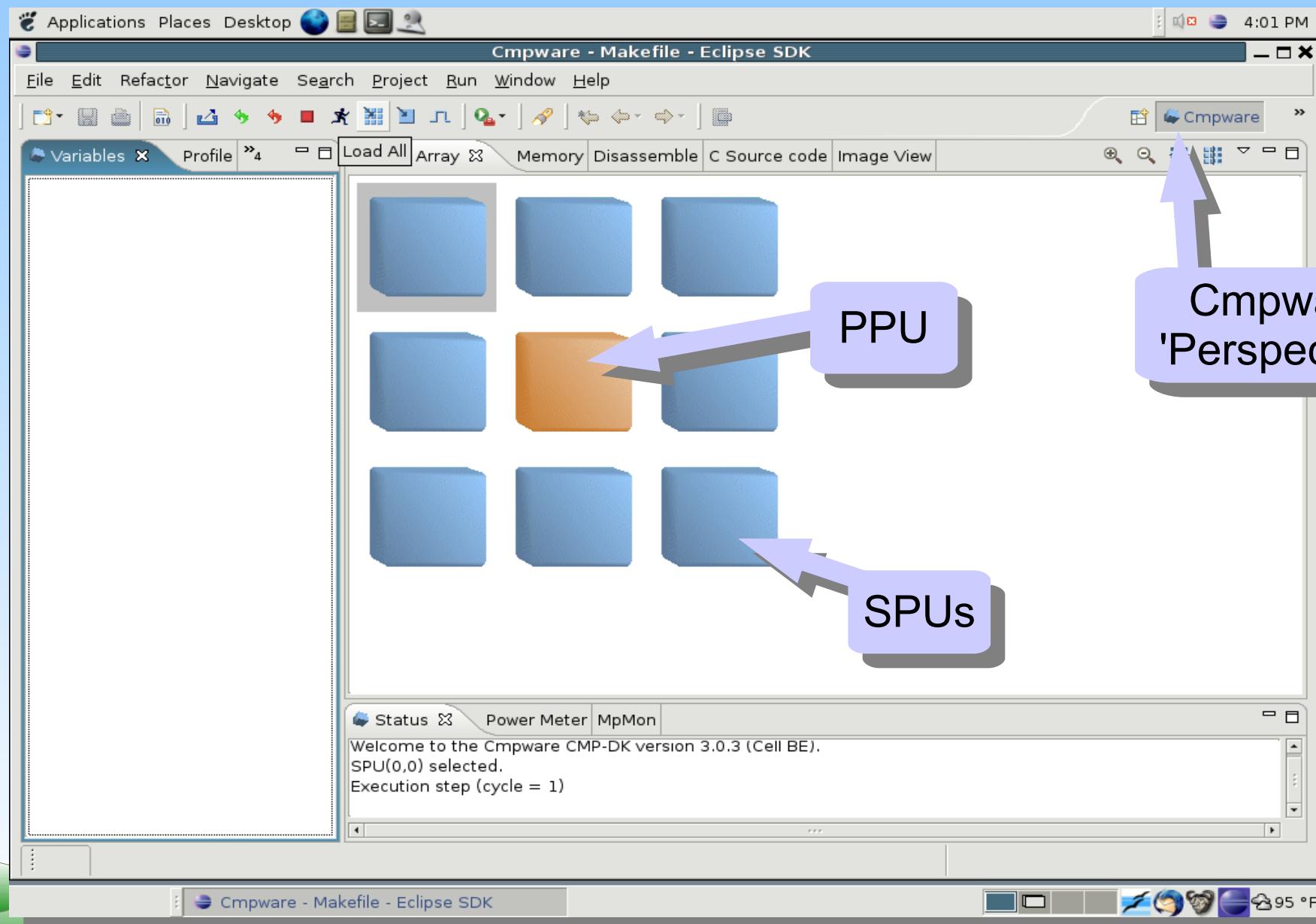
The left sidebar shows the C/C++ Projects view with files like Mandelbrot\_PPU.c, Mandelbrot\_SPU.asm, Mbmb\_opt\_spu.asm, etc. The right sidebar shows the C/C++ perspective with various tool icons and a list of recent projects.

Eclipse  
CDT (C/C++)  
'Perspective'

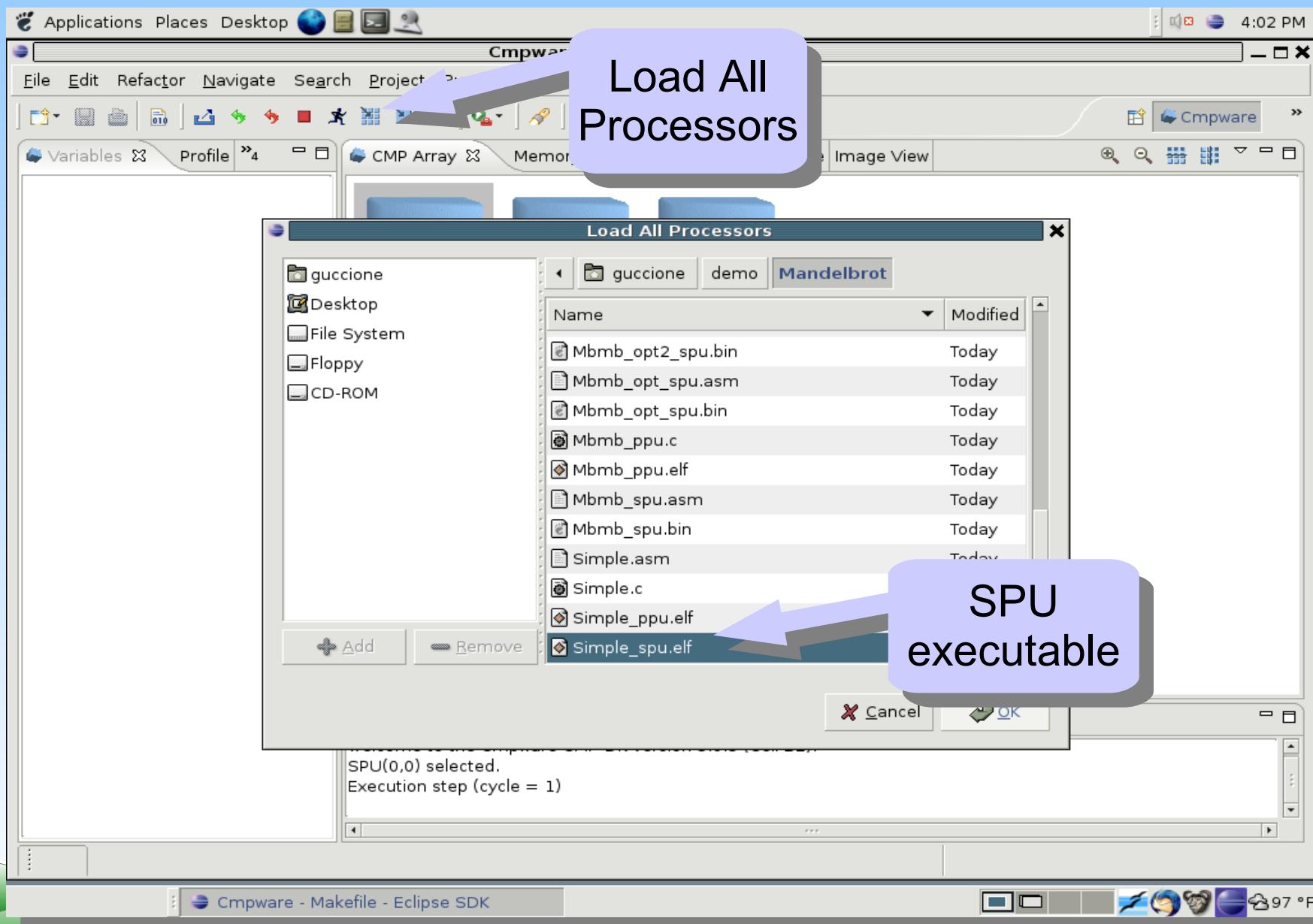
Makefile

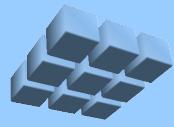


# Application I: The Main Display

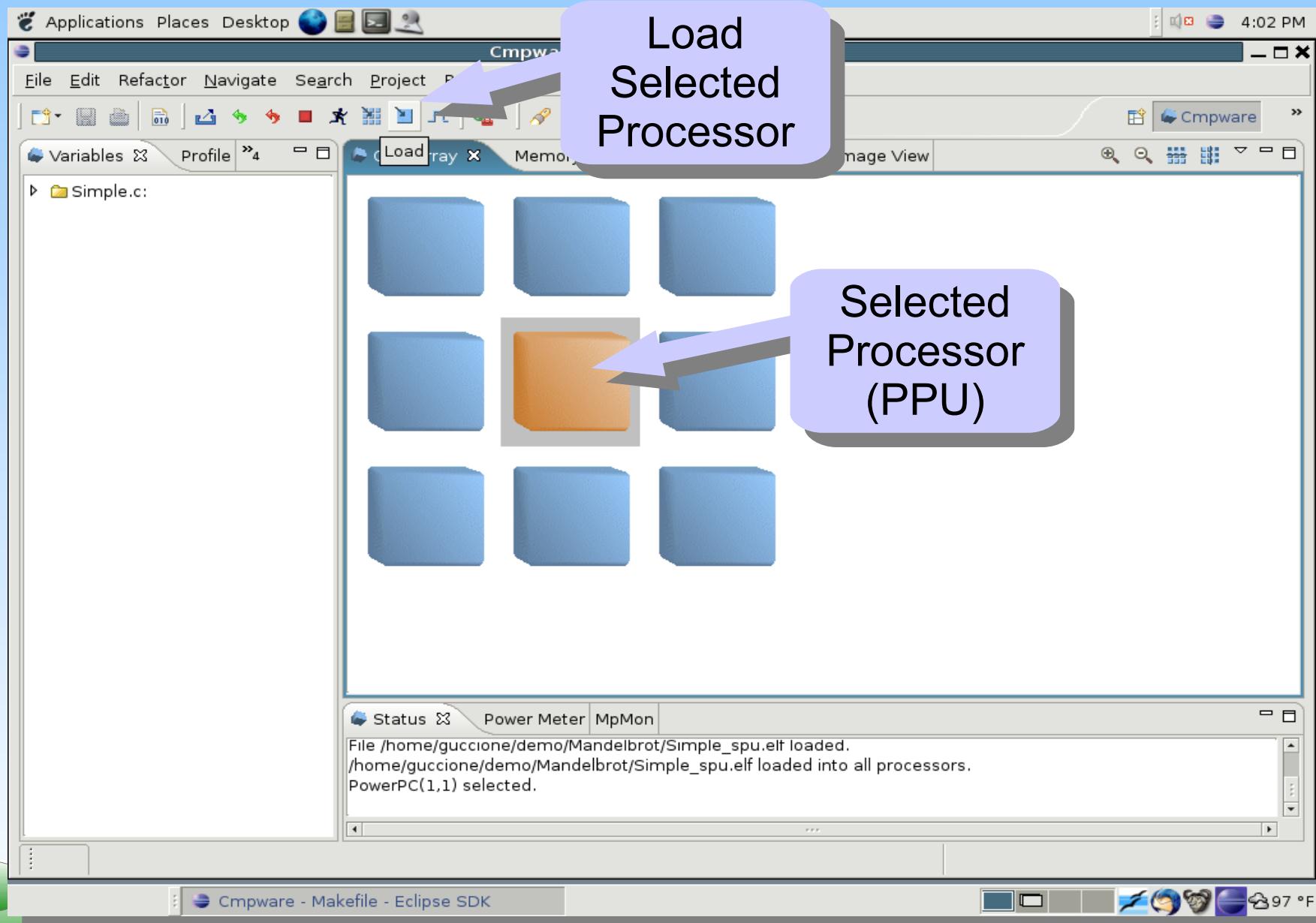


# Application I: Loading SPU Code

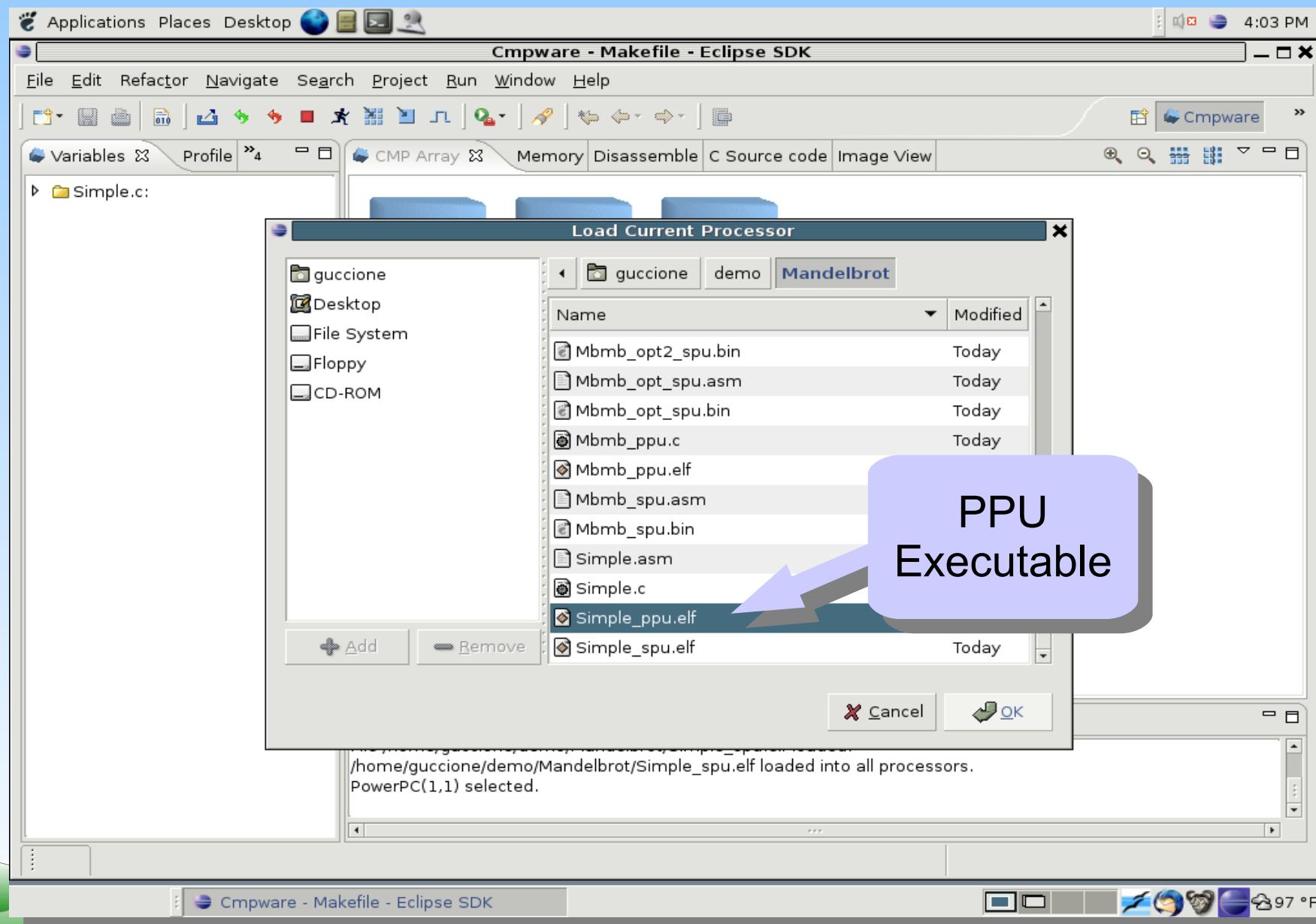


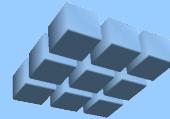


# Application I: Loading PPU Code

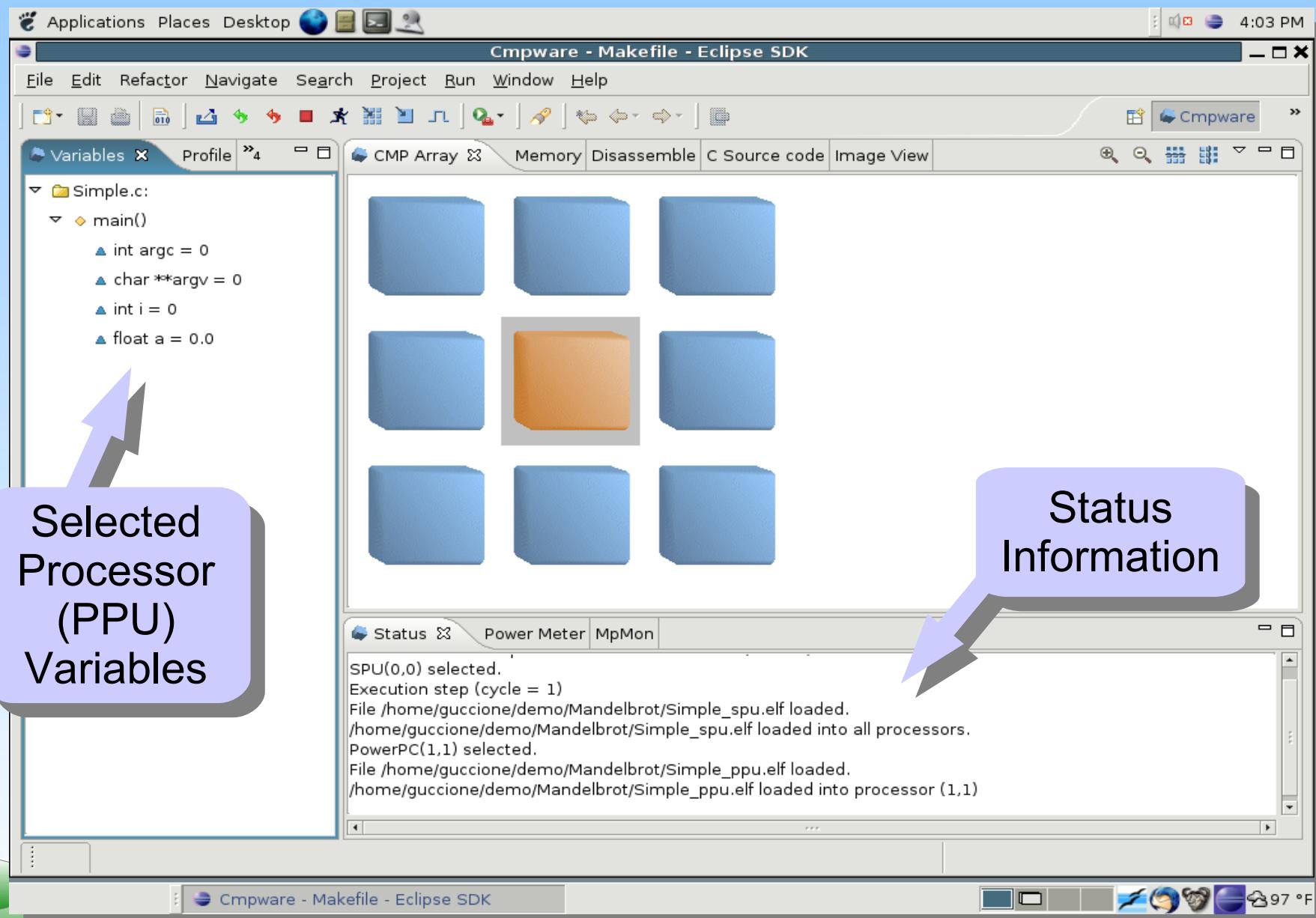


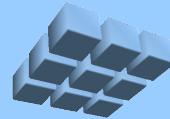
# Application I: Loading PPU Code



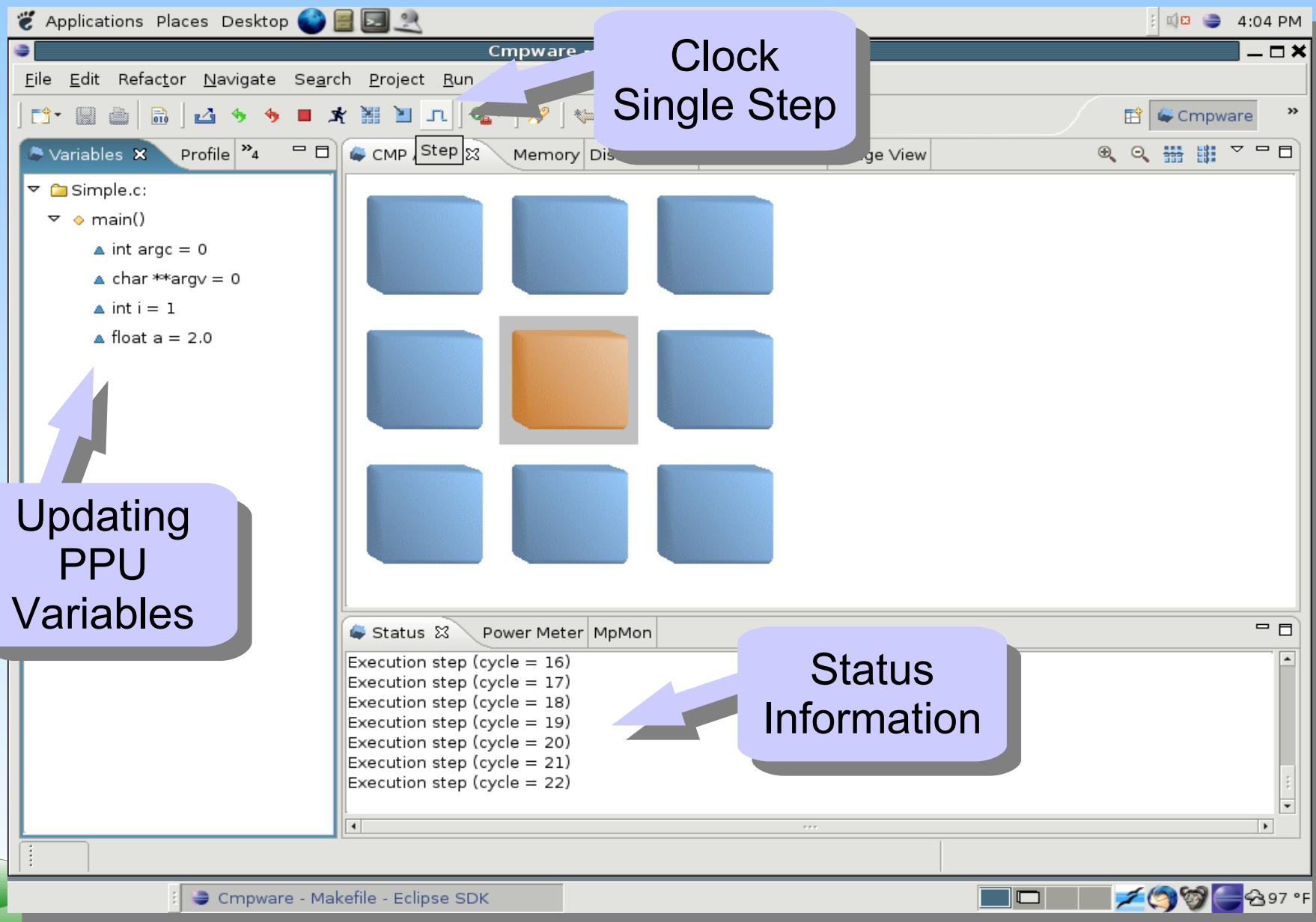


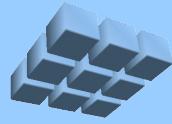
# Application I: All Code Loaded



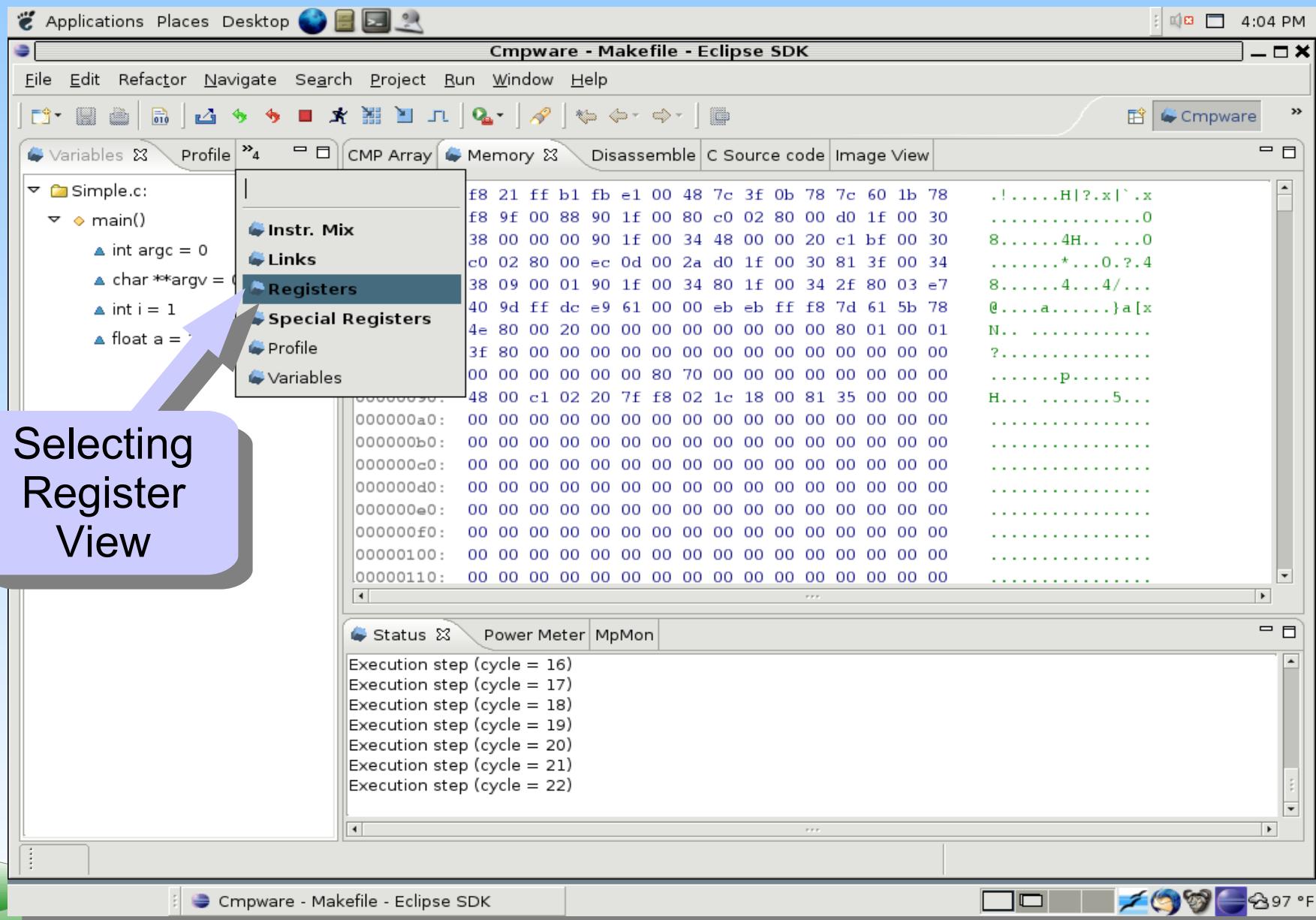


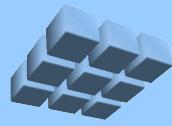
# Application I: Executing Code





# Application I: Memory View





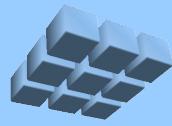
# Application I: Disassemble View

The screenshot shows the Cmpware - Makefile - Eclipse SDK application window. The title bar reads "Cmpware - Makefile - Eclipse SDK". The menu bar includes File, Edit, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and project management. The main workspace has several tabs: Registers, CMP Array, Memory, Disassemble (selected), C Source code, and Image View. A tooltip "Display performance data" is visible near the top right. The Registers view shows a table of general-purpose registers (r0 to r9) with their names, values, and addresses. The Disassemble view displays assembly code with addresses from 00000024 to 00000068. The assembly code includes instructions like stw, b, lfs, fadds, stfs, lwz, addi, stw, cmpi, bc, ld, or, and bclr. A purple callout bubble points to the Disassemble tab with the text "Disassemble View (PPU Selected)". Another purple callout bubble points to the Registers view with the text "Register View". The status bar at the bottom shows "Cmpware - Makefile - Eclipse SDK" and system information including a power meter and temperature (97 °F).

r[n]	Name	Value
0	r0.0	0
1	r0.1	1
2	r1.0	0
3	r1.1	1015712
4	r2.0	0
5	r2.1	32880
6	r3.0	0
7	r3.1	0
8	r4.0	0
9	r4.1	0
10	r5.0	0
11	r5.1	0
12	r6.0	0
13	r6.1	
14	r7.0	
15	r7.1	
16	r8.0	
17	r8.1	
18	r9.0	0
19	r9.1	0

00000024: 901f0034 stw r0, r31, 52  
00000028: 48000020 b 8  
0000002c: c1bf0030 lfs f13, r31, 48  
00000030: c0028000 lfs f0, r2, -32768  
00000034: ec0d002a fadds f0, f13, f0  
00000038: d01f0030 stfs f0, r31, 48  
0000003c: 813f0034 lwz r9, r31, 52  
00000040: 38090001 addi r0, r9, 1  
00000044: 901f0034 stw r0, r31, 52  
00000048: 801f0034 lwz r0, r31, 52  
0000004c: 2f8003e7 cmpi 7, r0, 999  
00000050: 409dffdc bc 4, 29, -9  
00000054: e9610000 ld r11, r1, 0  
00000058: ebefbff8 ld r31, r11, -8  
0000005c: 7d615b78 or r11, r1, r11  
00000060: 4e800020 bclr 20, 0, 8  
00000064: 00000000  
00000068: 00000000

step (cycle = 17)  
step (cycle = 18)  
step (cycle = 19)  
step (cycle = 20)  
step (cycle = 21)  
Execution step (cycle = 22)  
Execution step (cycle = 23)



# Application I: Source Code View

The screenshot shows the Cmpware - Makefile - Eclipse SDK interface. The main window has tabs for Registers, CMP Array, Memory, Disassemble, C Source code, and Image View. The C Source code tab is selected, displaying the following code:

```
/*
** A simple test program
**
** Copyright (c) 2007 Cmpware, Inc. All Rights Reserved.
**
*/
int main(int argc, char *argv[]) {
    int i;
    float a = 1.0;

    for(i=0; i<1000; i++)
        a = a + 1.0;

} /* end main() */


```

A callout bubble points to the Source Code View with the text "Source Code View (PPU Selected)".

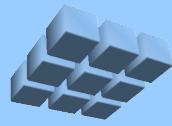
A callout bubble points to the Registers view with the text "Utilization / Power Meter".

The Registers view shows the following table:

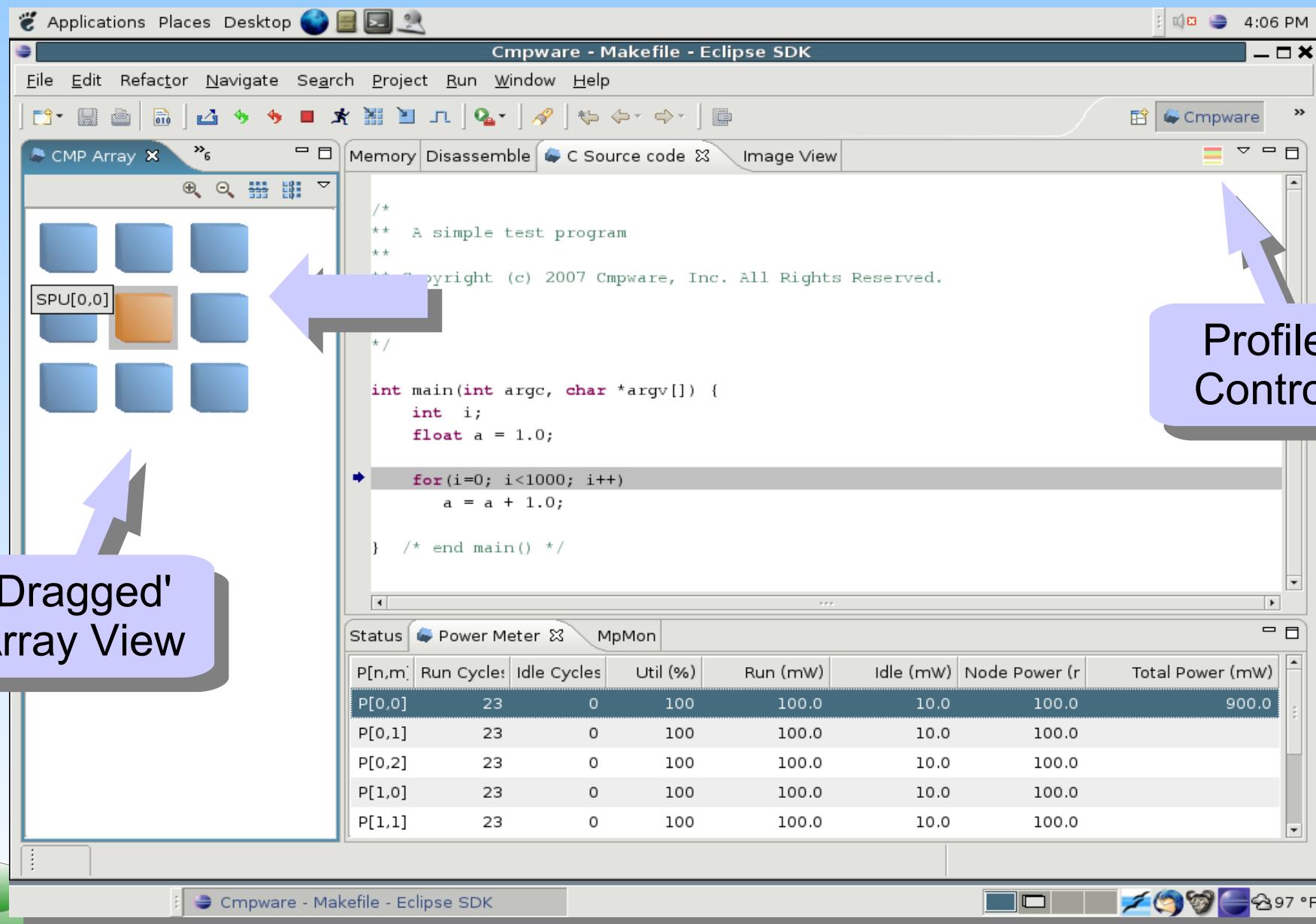
r[n]	Name	Value
0	r0.0	0
1	r0.1	1
2	r1.0	0
3	r1.1	f7fa0
4	r2.0	0
5	r2.1	8070
6	r3.0	0
7	r3.1	0
8	r4.0	0
9	r4.1	0
10	r5.0	0
11	r5.1	0
12	r6.0	0
13	r6.1	0
14	r7.0	0
15		
16		
17		
18		
19		

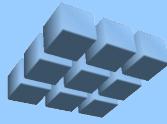
The Power Meter view shows the following data:

PPU	Run Cycles	Idle Cycles	Util (%)	Run (mW)	Idle (mW)	Node Power (mW)	Total Power (mW)
P[0,0]	23	0	100	100.0	10.0	100.0	900.0
P[0,1]	23	0	100	100.0	10.0	100.0	
P[0,2]	23	0	100	100.0	10.0	100.0	
P[1,0]	23	0	100	100.0	10.0	100.0	
P[1,1]	23	0	100	100.0	10.0	100.0	



# Application I: Moving Windows





# Application I: Source Profiling

The screenshot shows the Cmpware - Makefile - Eclipse SDK interface. On the left, there's a 'CMP Array' window displaying a 3x3 grid of memory blocks, with one central block highlighted in orange. The main workspace contains a C source code editor with the following code:

```
/*
** A simple test program
**
** Copyright (c) 2007 Cmpware, Inc. All Rights Reserved.
**
*/
int main(int argc, char *argv[]) {
    int i;
    float a = 1.0;

    for(i=0; i<1000; i++)
        a = a + 1.0;

} /* end main() */
```

A callout bubble labeled "PPU Profile Information" points to the code editor area, specifically highlighting the loop iteration. Another callout bubble labeled "37 Cycles of Execution" points to a table below, which shows execution counts for different memory locations.

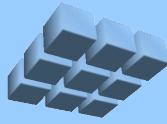
**Profiling Color Code**

Red	80-100%
Orange	60-80%
Yellow	40-60%
Green	20-40%
White	0-20%

**PPU Profile Information**

P[n,m]	Run Cycles	Idle Cycles	Util (%)	Run (mW)	Idle (mW)	Node Power (r)	Total Power (mW)
P[0,0]	37	0	100	100.0	10.0	100.0	900.0
P[0,1]	37	0	100	100.0	10.0	100.0	
P[0,2]	37	0	100	100.0	10.0	100.0	
P[1,0]	37	0	100	100.0	10.0	100.0	
P[1,1]	37	0	100	100.0	10.0	100.0	

**37 Cycles of Execution**



# Application I: Source Profiling

The screenshot shows the Cmpware - Makefile - Eclipse SDK interface. On the left, there's a 'CMP Array' view showing a 3x3 grid of colored squares (blue, orange, yellow). The main window has tabs for Memory, Disassemble, C Source code (which is selected), and Image View. The C Source code tab displays the following code:

```
/*
** A simple test program
**
** Copyright (c) 2007 Cmpware, Inc. All Rights Reserved.
**
*/
int main(int argc, char *argv[]) {
    int i;
    float a = 1.0;

    for(i=0; i<1000; i++)
        a = a + 1.0;

} /* end main() */
```

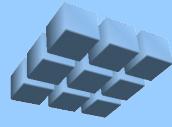
A purple callout bubble points to the inner loop of the code with the text "PPU Inner Loop 'Hot Spot'". Below the code, a horizontal bar indicates execution coverage or power usage, with a red section underlining the loop body.

**Profiling Color Code**

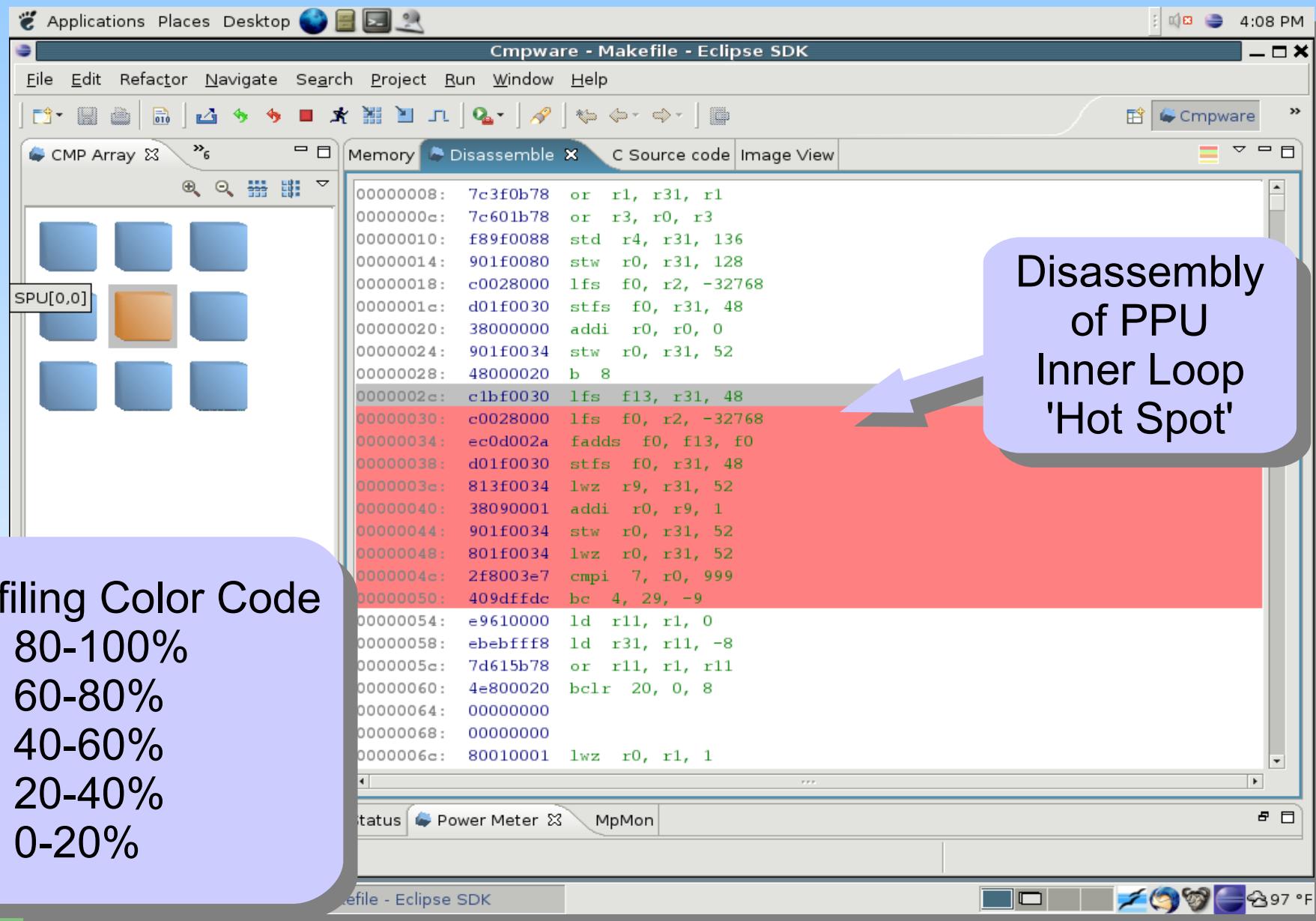
Red	80-100%
Orange	60-80%
Yellow	40-60%
Green	20-40%
White	0-20%

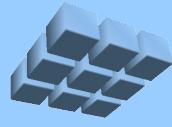
A purple callout bubble points to the 'Power Meter' table with the text "64 Cycles of Execution". The table shows the following data:

P[n,m]	Run Cycles	Idle Cycles	Util (%)	Run (mW)	Idle (mW)	Node Power (r)	Total Power (mW)
P[0,0]	64	0	100	100.0	10.0	100.0	900.0
P[0,1]	64	0	100	100.0	10.0	100.0	
P[0,2]	64	0	100	100.0	10.0	100.0	
P[1,0]	64	0	100	100.0	10.0	100.0	
P[1,1]	64	0	100	100.0	10.0	100.0	

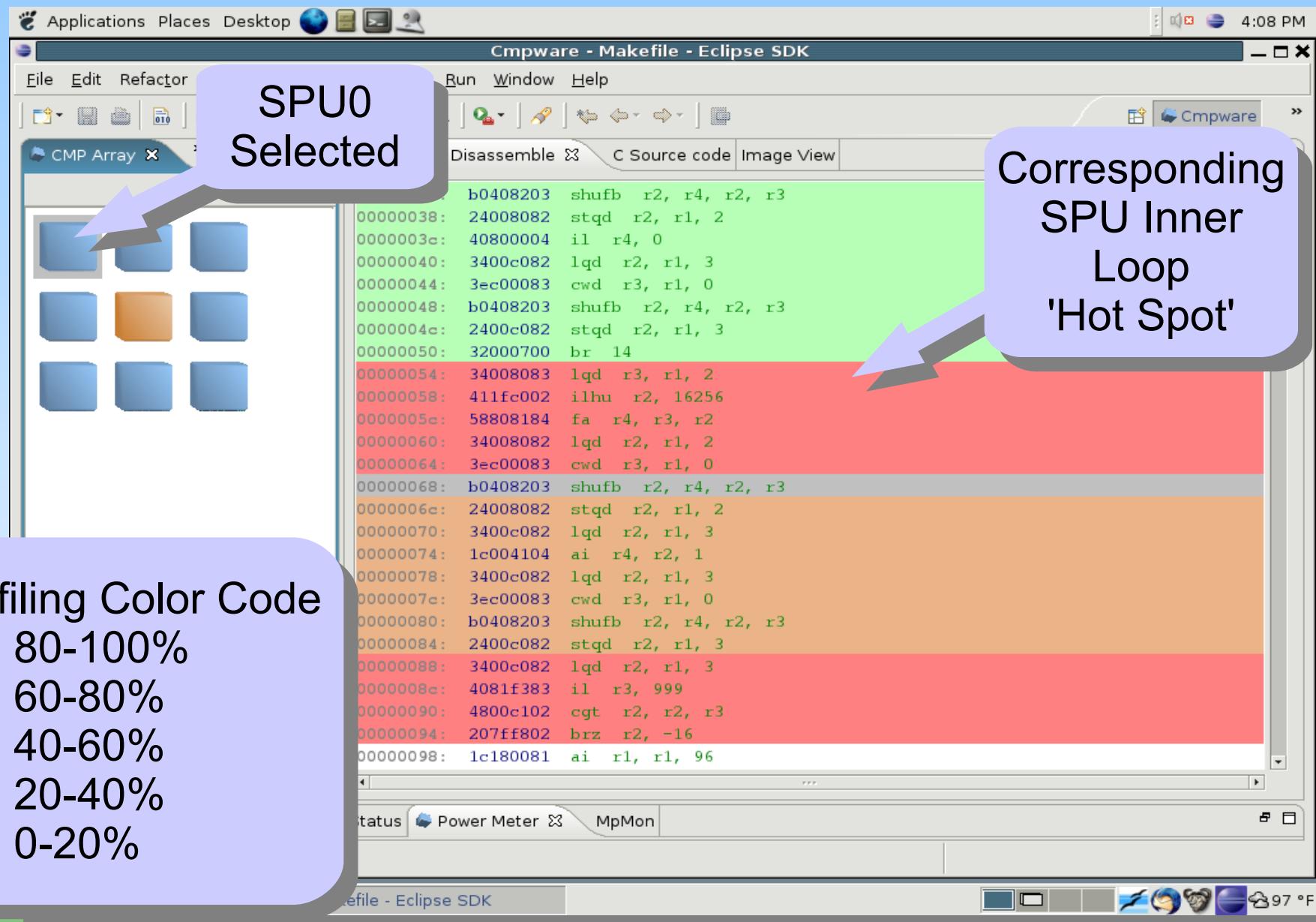


# Application I: Assembly Profiling



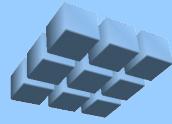


# Application I: Assembly Profiling

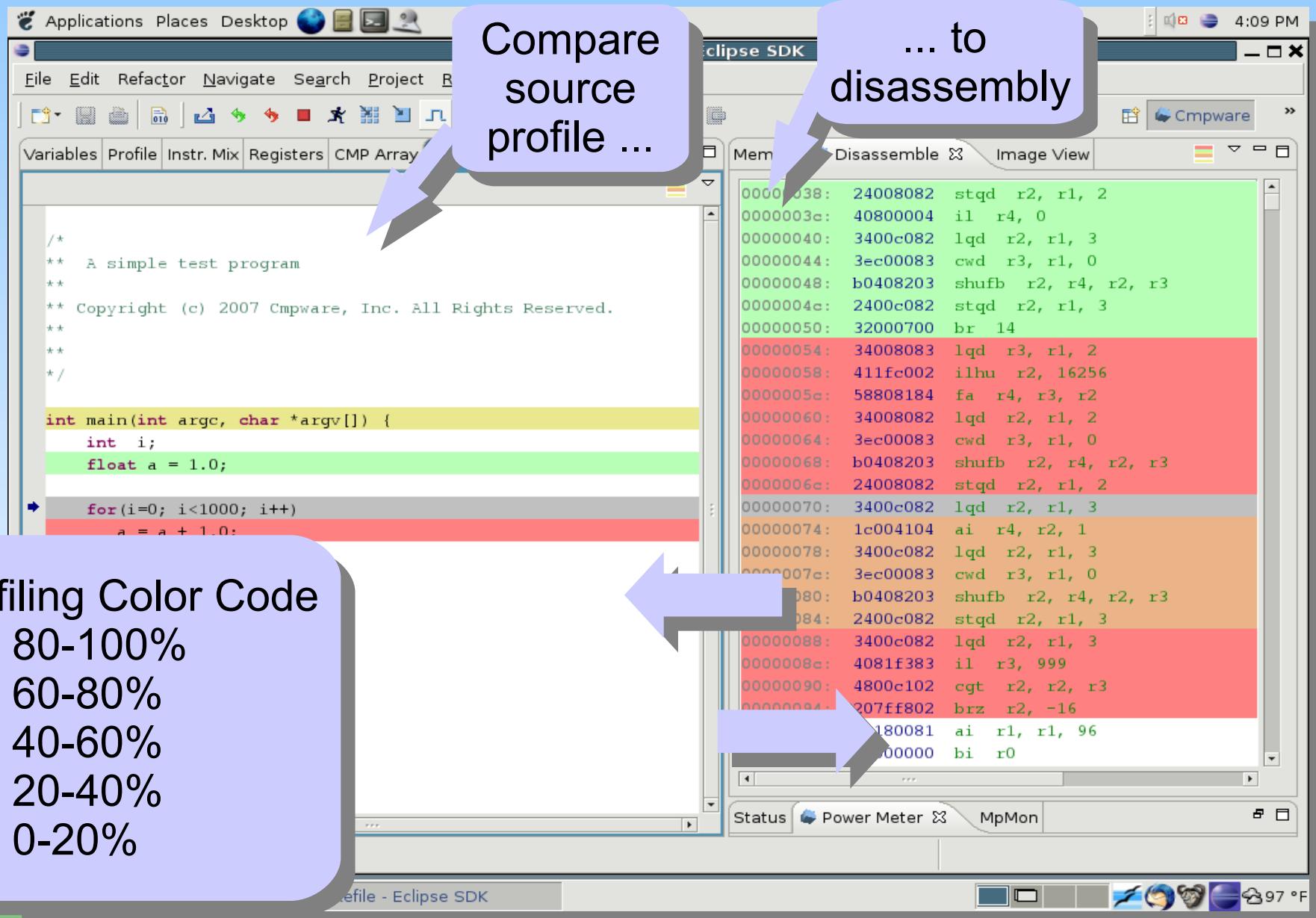


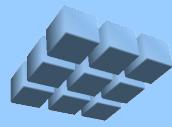
Profiling Color Code

- █ 80-100%
- █ 60-80%
- █ 40-60%
- █ 20-40%
- █ 0-20%

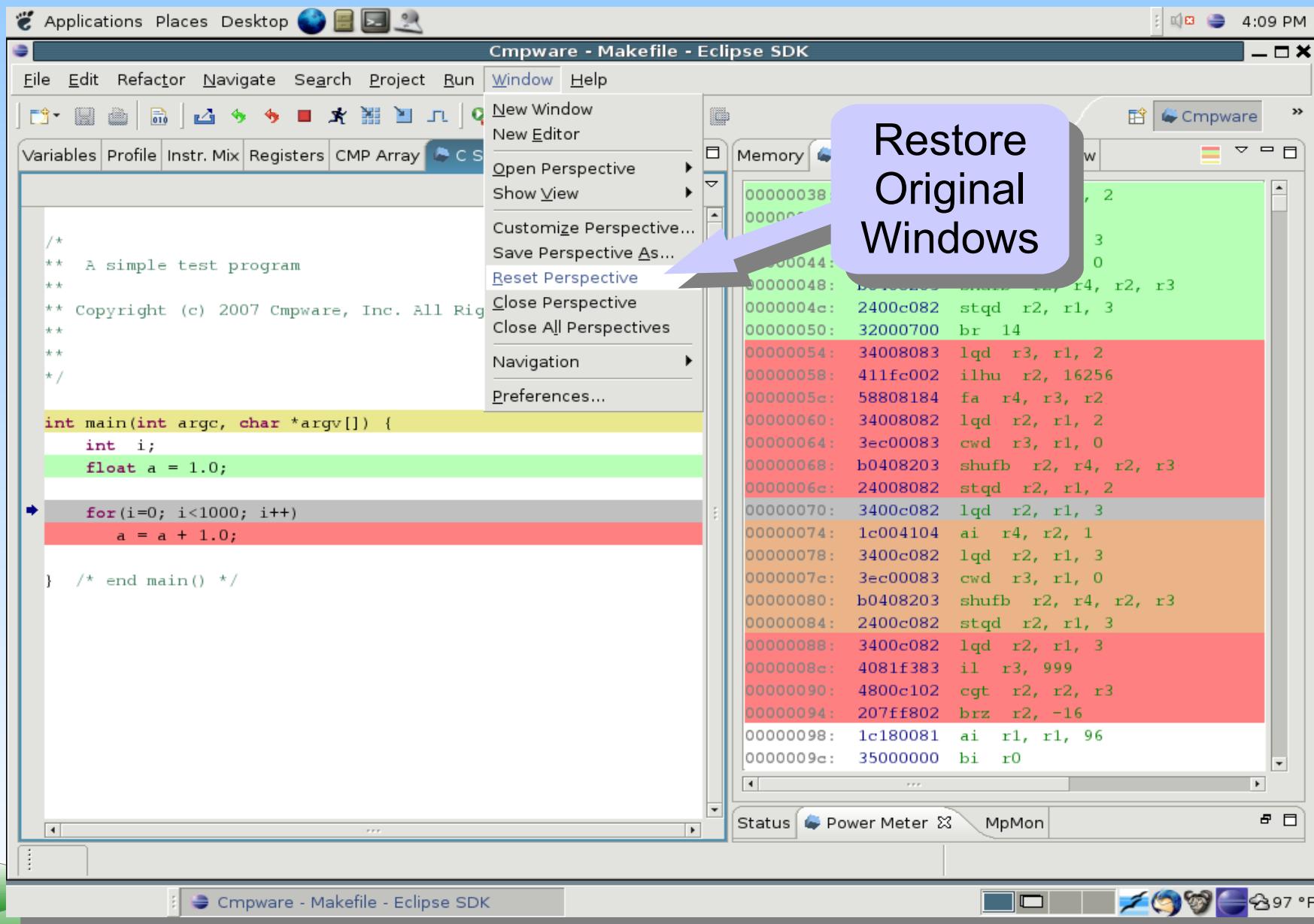


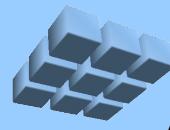
# Application I: Moving Windows



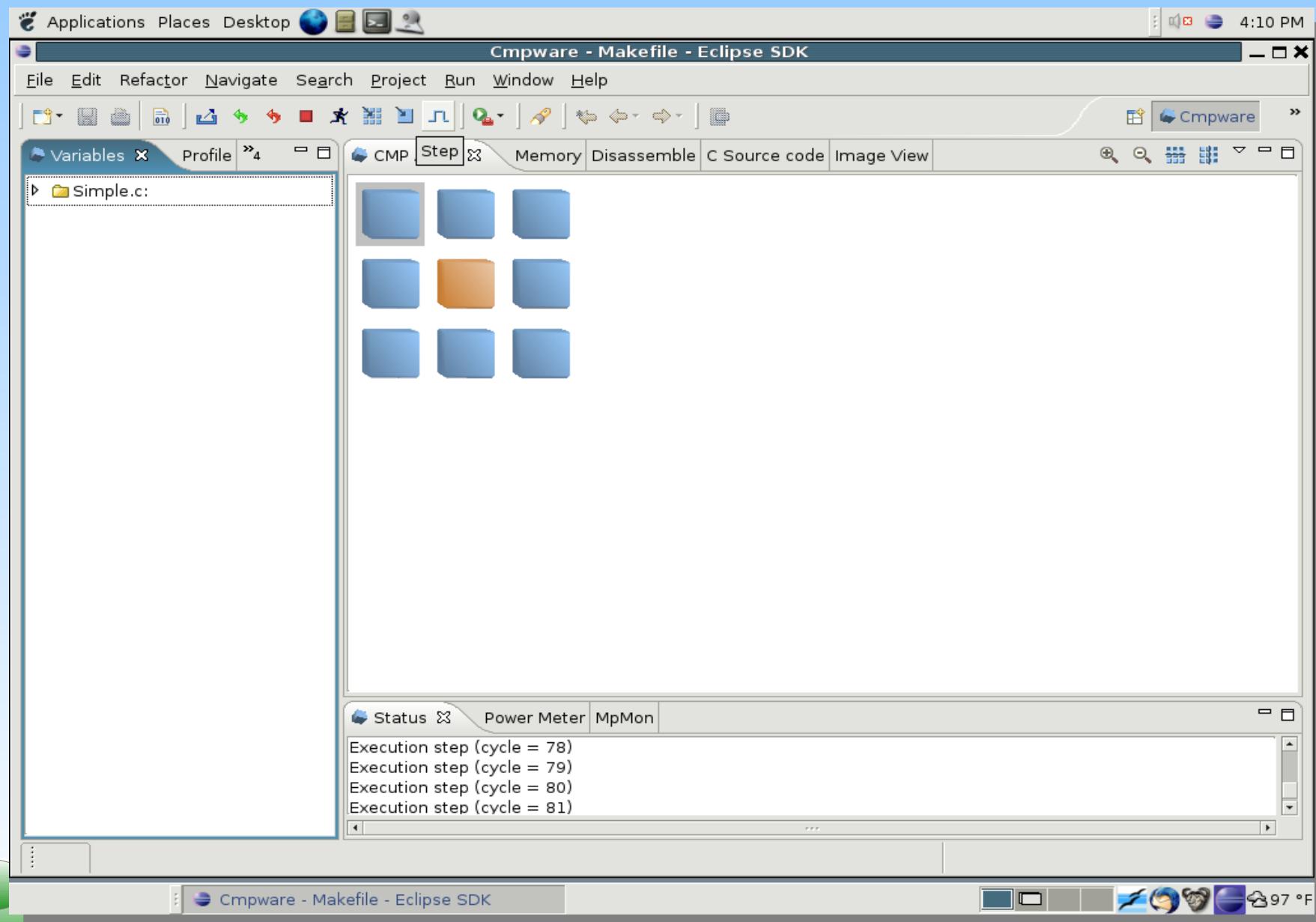


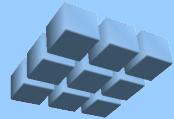
# Application I: Reset Perspective





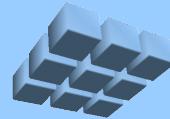
# Application I: Default Perspective

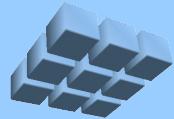




# Application I: Overview

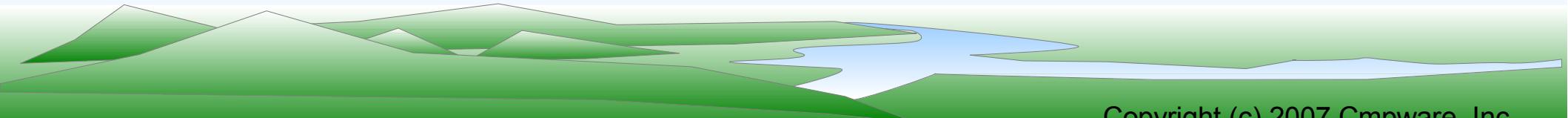
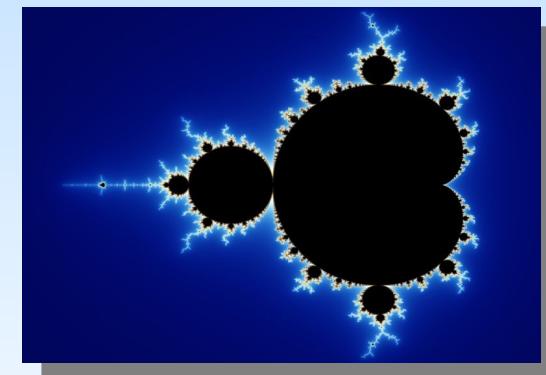
- Introduces the *Cmpware CMP-DK*
- Loading and execution of *Cell BE* code
- Interactive graphical profiler
- Data displays (memory, registers, etc.)
- Code displays (source, disassembly)
- Changing views (moving, resizing)
- Interacting with other *Eclipse* plugins (CDT)

- 
- I. Introduction
  - II. Installing the *Cmpware CMP-DK*
  - III. Application I: A Simple Program
  - IV. Application II: Shared Memory
  - V. Application III: Mailboxes
  - VI. Optimization I: SPU Dependencies
  - VII. Optimization II: SPU Dual Issue
  - VIII. System Level Analysis
  - IX. Overview

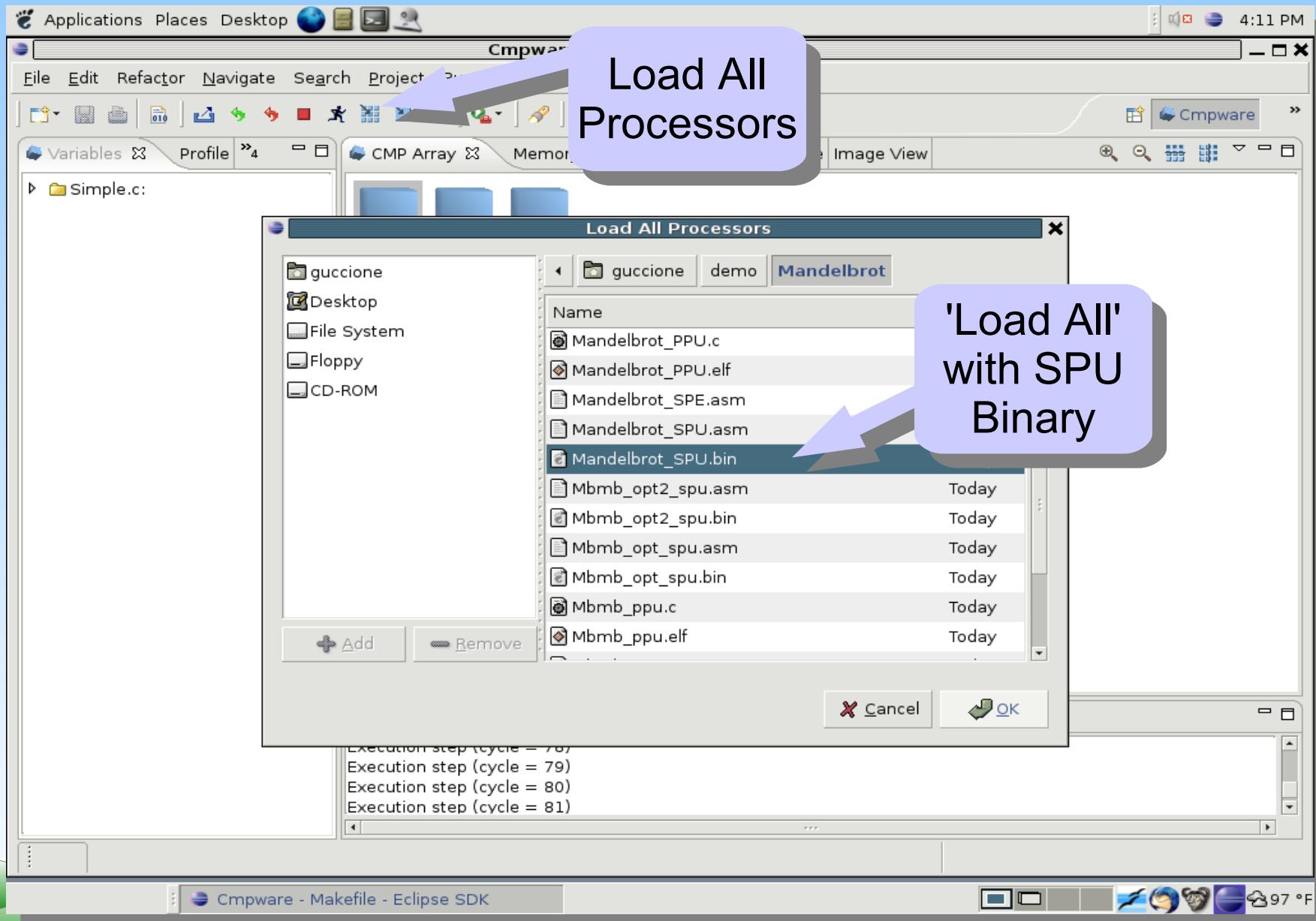


# Application II: Shared Memory

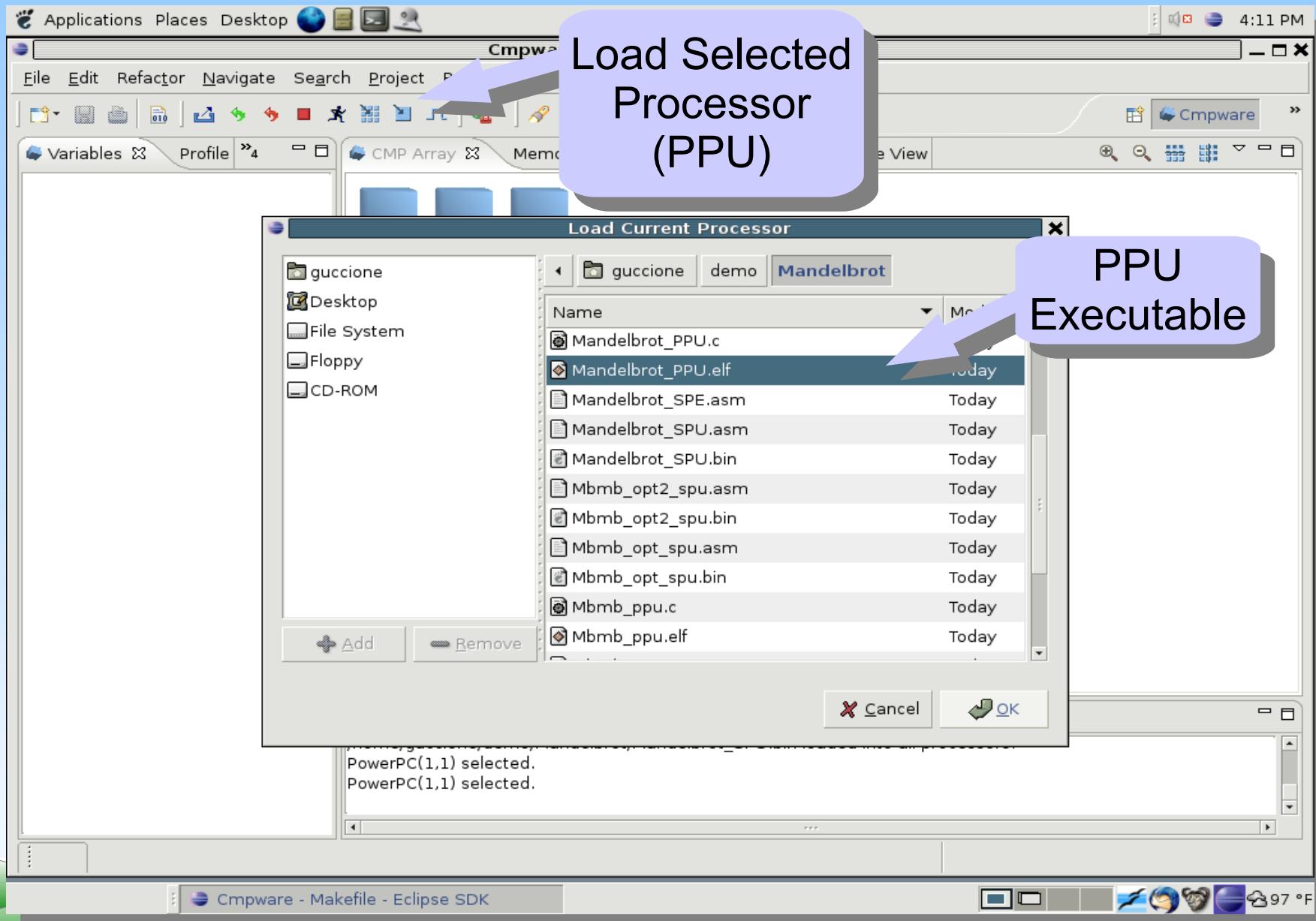
- Application II: the Mandelbrot Set
- All communication via shared memory
- Floating point intensive, highly parallel
- Demonstrates *Cell BE* architecture
- PPU controls SPUs
- All work done on SPUs
- SPUs coded in assembly

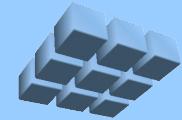


# Application II: Load SPU Code

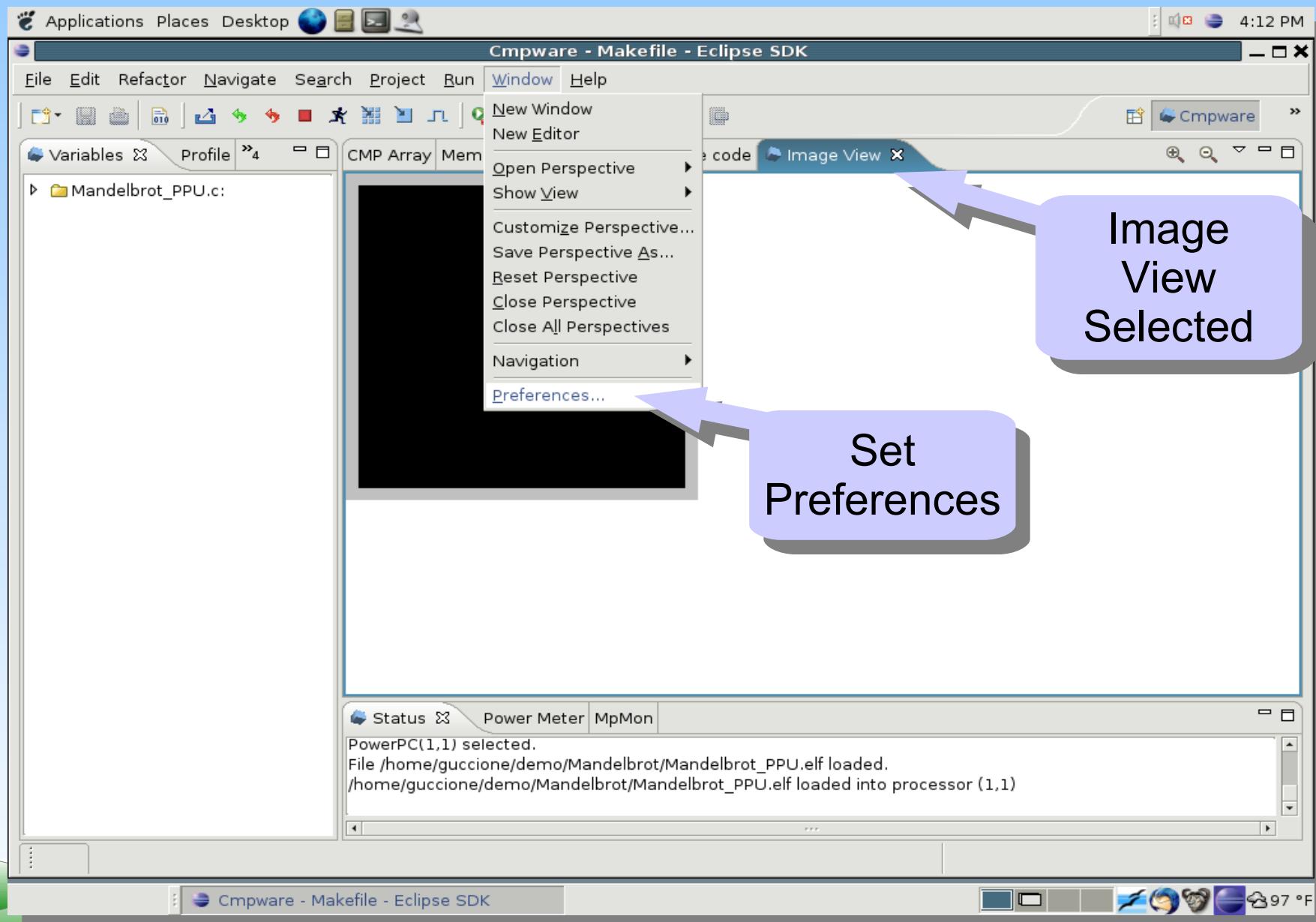


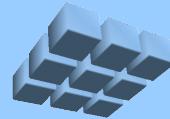
# Application II: Load PPU Code



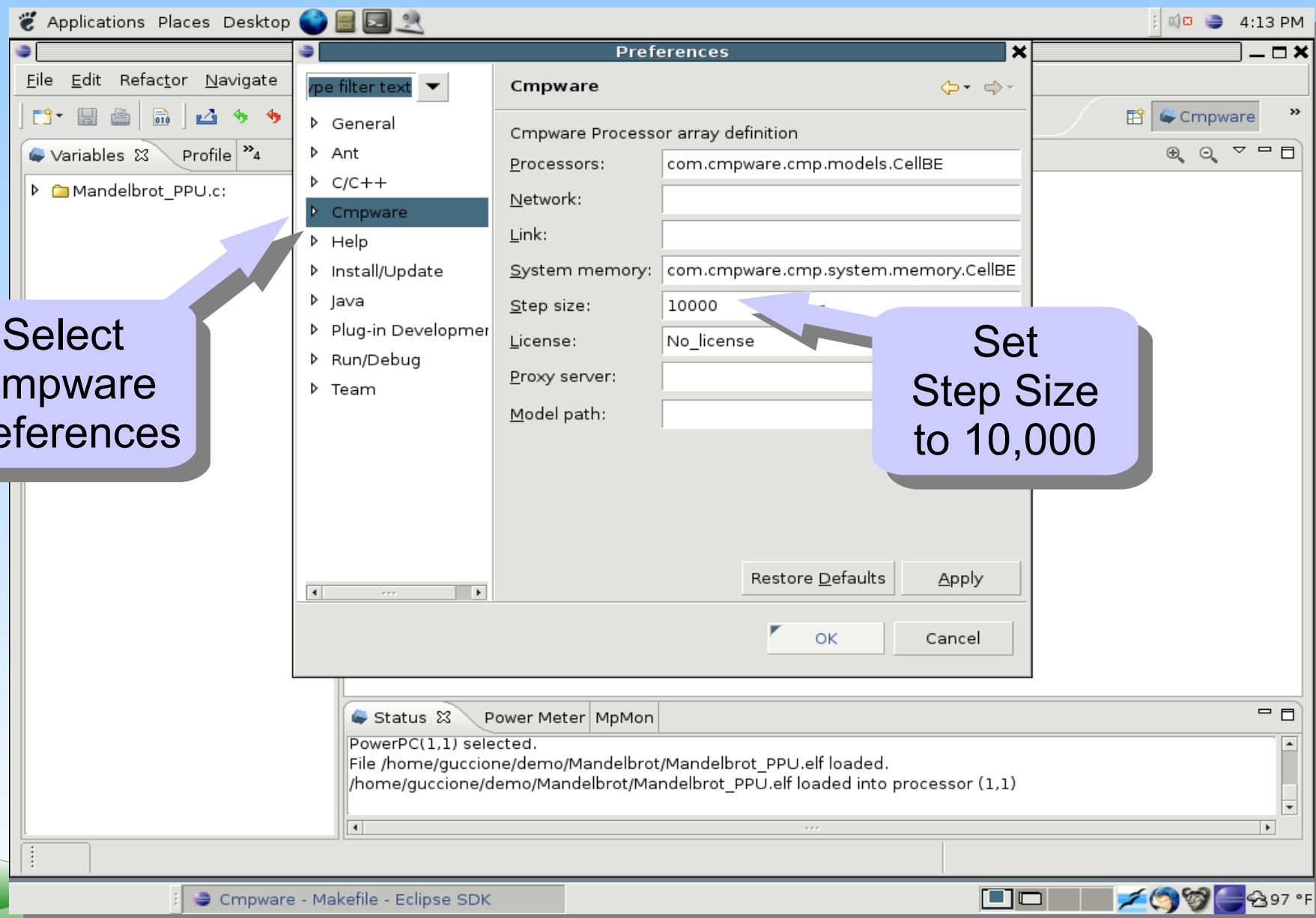


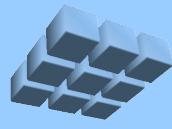
# Application II: Set Preferences



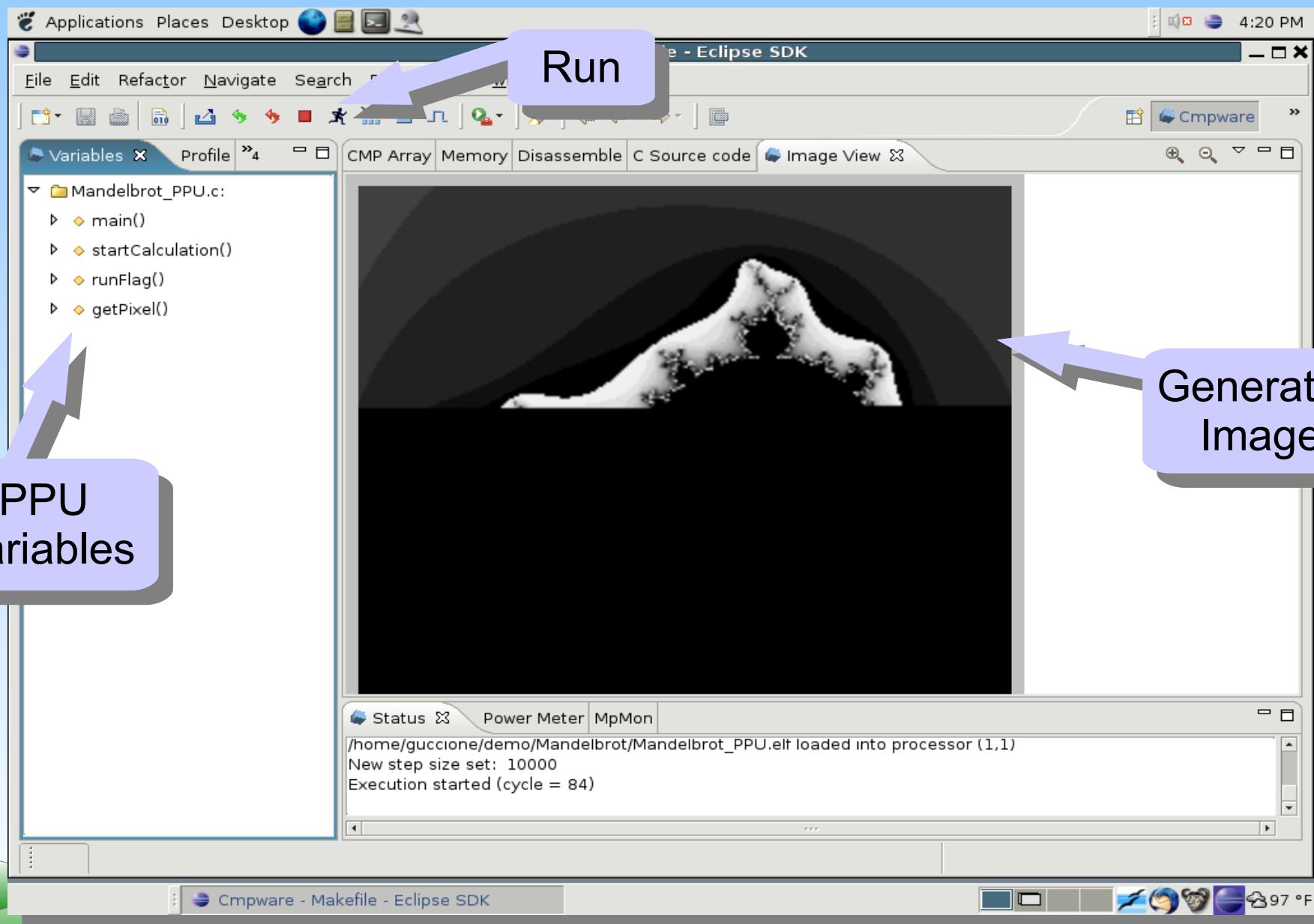


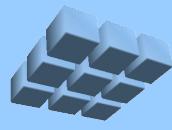
# Application II: Set Step Size



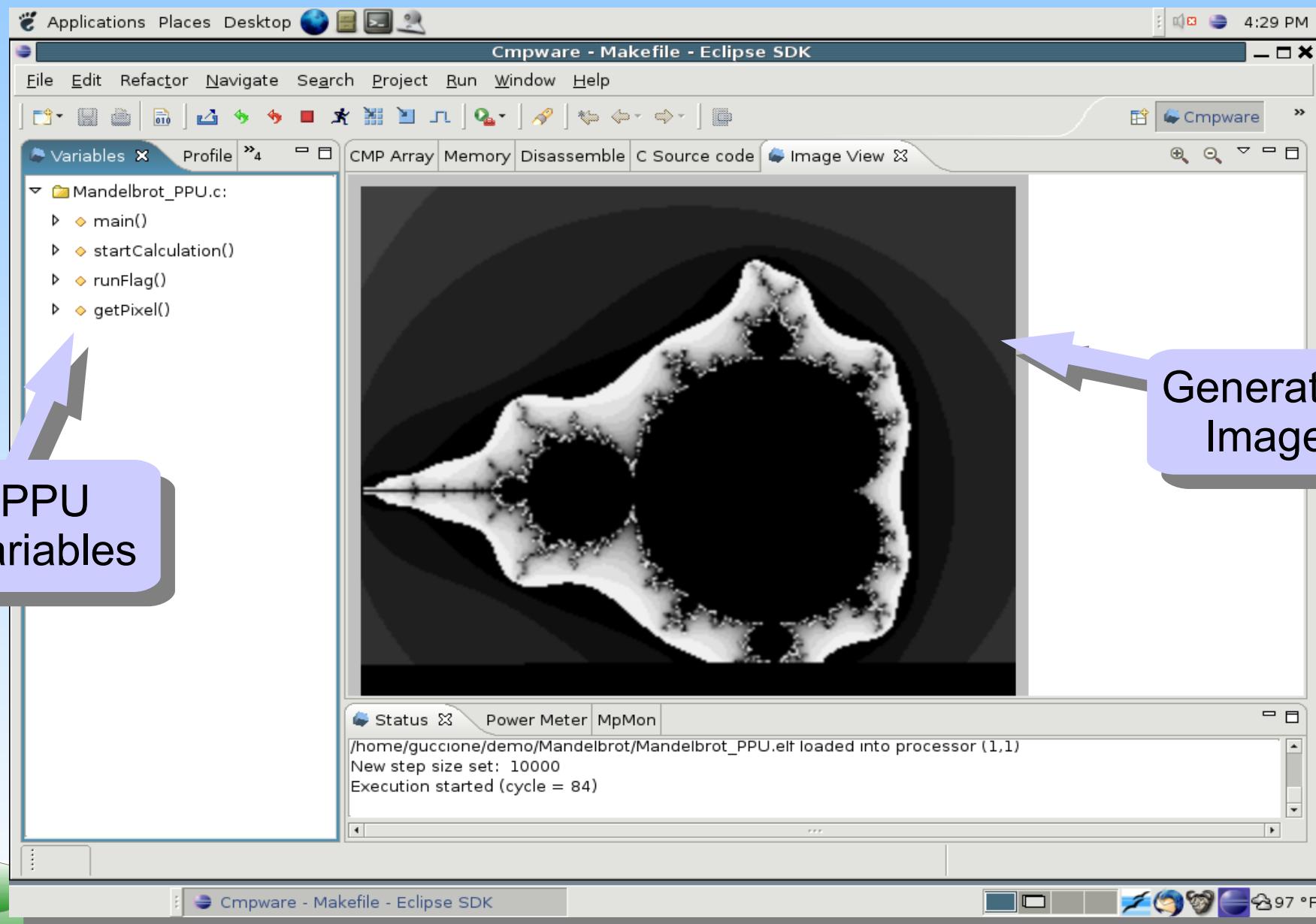


# Application II: Running the Code





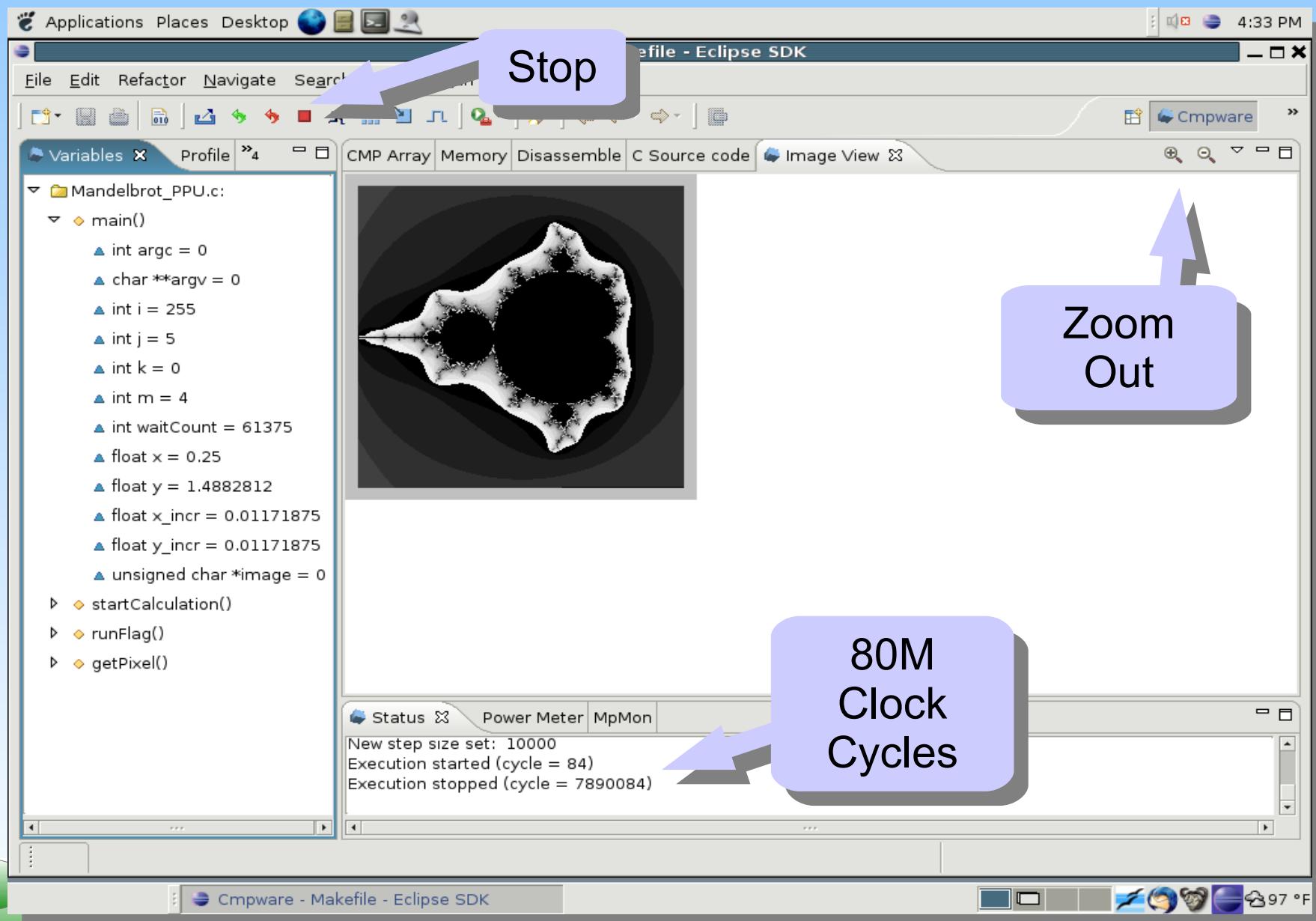
# Application II: Running the Code

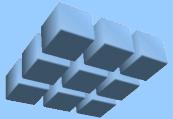


PPU  
Variables

Generated  
Image

# Application II: Stopping Execution

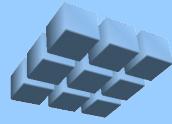




# SPU Assembly Code

```
--  
-- This is the inner loop of the  
-- Mandelbrot algorithm for the  
-- CellBE SPU. It is used to  
-- generate the data used by  
-- Mandlebrot_PPE.c  
  
-- Copyright (c) 2007 Cmpware, Inc.  
-- All Rights Reserved.  
  
-- Useful constants  
#define zero r0  
#define one r1  
  
-- The (shared memory) parameters  
#define flag r8  
#define x r9  
#define y r10  
#define cutoff r11  
#define imax r12  
#define pixel r13  
  
-- Other variables  
#define params r14  
#define tmp0 r15  
#define tmp1 r16  
#define tmp2 r17  
#define tmp3 r18
```

```
#define z_re r20  
#define z_im r21  
#define c_re r22  
#define c_im r23  
#define done_mask r24  
#define icount r25  
  
-- Initialize constants  
il zero, 0  
il one, 1  
il done_mask, 0  
il icount, 0  
il params, 0x1000  
  
-- Wait for the 'go' flag  
lqx flag, params, zero  
brz flag, -1  
  
-- Load parameters  
il tmp0, 16  
lqx x, params, tmp0  
il tmp0, 32  
lqx y, params, tmp0  
il tmp0, 48  
lqx cutoff, params, tmp0  
il tmp0, 64  
lqx imax, params, tmp0
```



# SPU Assembly Code (cont.)

```
-- Load Z and C initial values
a    z_re, zero, zero
a    z_im, zero, zero
a    c_re, x, zero
a    c_im, y, zero

-- z^2 (re): (z.re * z.re) - (z.im * z.im)
fm   tmp1, z_re, z_re
fm   tmp2, z_im, z_im
fs   tmp3, tmp1, tmp2

--      z^2 (im): (z.re * z.im) +
--                  (z.re * z.im)
fm   tmp1, z_re, z_im
fa   z_im, tmp1, tmp1
fa   z_re, tmp3, zero

-- z = z^2 + c
fa   z_re, z_re, c_re
fa   z_im, z_im, c_im

-- Is ((z.re^2) + (z.im^2)) > cutoff
fm   tmp2, z_re, z_re
fm   tmp3, z_im, z_im
fa   tmp2, tmp3, tmp2
fcgt  tmp3, tmp2, cutoff
```

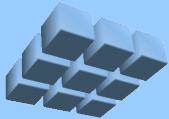
```
-- Increment iteration count for values
-- still less than cutoff
or   done_mask, done_mask, tmp3
and  tmp2, done_mask, one
a    icount, icount, tmp2

-- imax = imax - 1
sf   imax, one, imax
brnz imax, -16

-- Copy results to shared memory
il   tmp0, 80
stqx icount, params, tmp0

-- Set 'ready' flag
stqx zero, params, zero

-- Go back to start
-- (and wait for another request)
bra  0
nop
nop
```



# PPU 'C' Code

```
/*
** This program is used to generate
** the Mandelbrot set. For higher
** resolution, increase the ITERATIONS
** to several thousand. This will greatly
** increase the computation time, but is
** necessary for zooming in to higher
** resolutions. The other constants control
** the region of the Mandlebrot set being
** mapped.
**
** This file is the PPU (PowerPC) portion
** of the code which runs on the PPU unit
** of the Cell BE. It uses shared memory
** to drive the computation taking place
** on the SPEs. The SPEs each calculate
** one set of four pixels in the Mandlebrot
** set and return the data via shared memory.
**
** Copyright (c) 2007 Cmpware, Inc.
** All rights reserved.
*/
#include "CellBE.h"

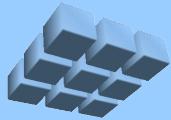
/* Functions */
unsigned char getPixel(int spe,
                      int pixelNum);
int runFlag(int spe);
void startCalculation(int spe, float x,
                      float y, float x_incr);
```

```
/*
** Constants
*/

#define ITERATIONS 100 // Default: 100
#define CUTOFF 5.0 // Default: 5.0
#define X_START -2.0 // Default: -2.0
#define X_END 1.0 // Default: 1.0
#define Y_START -1.5 // Default: -1.5
#define Y_END 1.5 // Default: 1.5
#define X_PIXELS 256
#define Y_PIXELS 256

#define SPU_READY 0
#define SPU_BUSY 1

/* The parameters for the SPE (quadrwords) */
typedef struct {
    int flag[4]; // The control flag
    float x[4]; // X (c.re)
    float y[4]; // Y (c.im)
    float cutoff[4]; // 'cutoff'
    int imax[4]; // Max iterations
    int pixel[4]; // Pixel data
} params;
```



# PPU 'C' Code (cont.)

```
int
main(int argc, char *argv[]) {
    int i, j, k, m;
    int waitCount = 0;
    float x = X_START;
    float y = Y_START;
    float x_incr = (X_END - X_START) /
        ((float) X_PIXELS);
    float y_incr = (Y_END - Y_START) /
        ((float) Y_PIXELS);
    unsigned char *image =
        (unsigned char *) 0x40000L;

    for (i=0; i<Y_PIXELS; i++) {
        for (j=0; j<(X_PIXELS/(SPUS*4)); j++) {

            /* Start calculations */
            for (k=0; k<SPUS; k++) {
                startCalculation(k, x, y, x_incr);
                x = x + (4 * x_incr);
            } /* end for(k) */
        }
    }
}
```

```
/* Get pixel results */
for (k=0; k<SPUS; k++) {
    while (runFlag(k) != SPU_READY)
        waitCount++;
    for (m=0; m<4; m++)
        *image++ = getPixel(k,m);
} /* end for(k) */

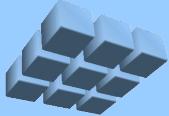
} /* end for(j) */

x = X_START;
y = y + y_incr;

} /* end for(i) */

for(;;) ;

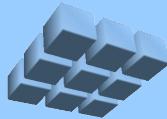
} /* end main() */
```



# PPU 'C' Code (cont.)

```
/**  
 * This function sends the required  
 * parameters to the SPE and  
 * sets the shared memory flag to  
 * begin the computation.  
 *  
 * @param spu the SPU number.  
 * Should be 0 to 7.  
 *  
 * @param x the x coordinate.  
 * This is the real portion  
 * of the complex number 'C'  
 * used in the calculation.  
 *  
 * @param y the y coordinate.  
 * This is the imaginary  
 * portion of the complex  
 * number 'C' used in the  
 * calculation.  
 *  
 * @param x_incr the x coordinate  
 * increment. This is the amount  
 * x is incremented per pixel.  
 */
```

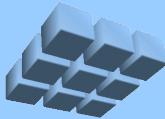
```
void startCalculation(int spu, float x,  
                      float y, float x_incr) {  
    params *p = (params *) (long) (BP_BASE +  
        (spu * SPU_MEMORY_RANGE) + 0x1000);  
  
    p->x[0] = x;  
    p->x[1] = x + x_incr;  
    p->x[2] = x + (2 * x_incr);  
    p->x[3] = x + (3 * x_incr);  
    p->y[0] = y;  
    p->y[1] = y;  
    p->y[2] = y;  
    p->y[3] = y;  
    p->cutoff[0] = CUTOFF;  
    p->cutoff[1] = CUTOFF;  
    p->cutoff[2] = CUTOFF;  
    p->cutoff[3] = CUTOFF;  
    p->imax[0] = ITERATIONS;  
    p->imax[1] = ITERATIONS;  
    p->imax[2] = ITERATIONS;  
    p->imax[3] = ITERATIONS;  
    p->flag[1] = 0;  
    p->flag[2] = 0;  
    p->flag[3] = 0;  
    p->flag[0] = SPU_BUSY; // Start  
} /* end startCalculation() */
```



# PPU 'C' Code (cont.)

```
/**  
 * This function returns the 'run'  
 * flag from shared memory. It  
 * should have a value of WAIT,  
 * GO or DONE.  
 */  
  
/** @param spu the SPU number.  
 * Should be 0 to 7.  
 */  
  
/** @returns the value of the SPE  
 * 'run' flag is returned.  
 */  
  
int runFlag(int spu) {  
    params *p = (params *) (long) (BP_BASE +  
        (spu * SPU_MEMORY_RANGE) + 0x1000);  
    return (p->flag[0]);  
} /* end runFlag() */
```

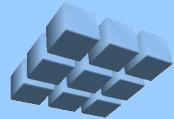
```
/**  
 * This function gets a pixel  
 * value from a PE. There are  
 * four pixels produced per  
 * calculation.  
 */  
  
/** @param spu the SPE number.  
 * Should be 0 to 7.  
 */  
  
/** @param pixelNum the pixel  
 * number. Should be 0 to 3.  
 */  
  
/** @returns A 32 bit value  
 * representing the number  
 * of iterations is  
 * returned. This value is  
 * always less than or  
 * equal to the 'cutoff'  
 * parameter.  
 */  
  
unsigned char getPixel(int spu,  
    int pixelNum) {  
    unsigned char pixel;  
    params *p = (params *) (long) (BP_BASE +  
        (spu * SPU_MEMORY_RANGE) + 0x1000);  
    pixel = (unsigned char)  
        ((p->pixel[pixelNum] & 0x0f) << 4);  
    return (pixel);  
} /* end getPixel() */
```



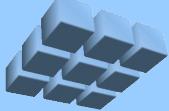
# Application II: Overview

- Introduces PPU / SPU communication
- Demonstrates shared memory programming
- Introduces *Eclipse 'Preferences'*
- Demonstrates Image View

*A Note on Synchronization:* This communication scheme eliminates the need for 'semaphores'. This is not generally true in shared memory software.

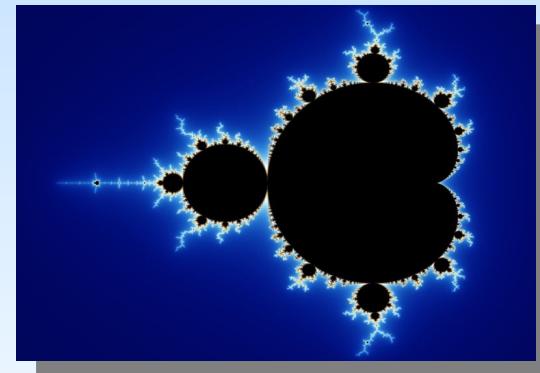


- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis
- IX. Overview

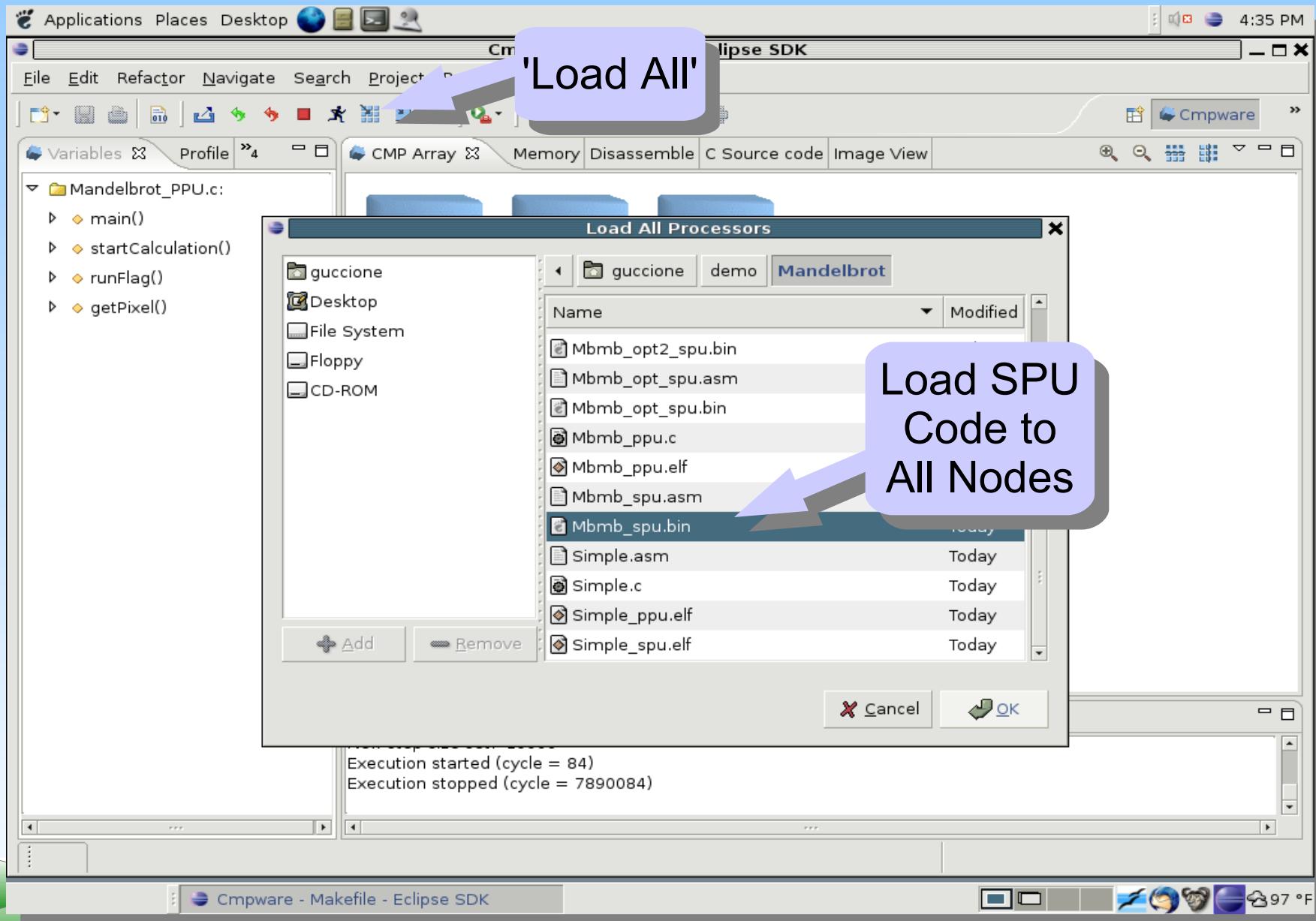


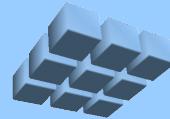
# Application III: Mailboxes

- SPU / PPU communication via 'mailboxes'
- Mandelbrot application converted from shared memory to mailboxes
- Mailboxes: 32-bit communication channels
  - Point to point
  - Synchronized
  - Blocking

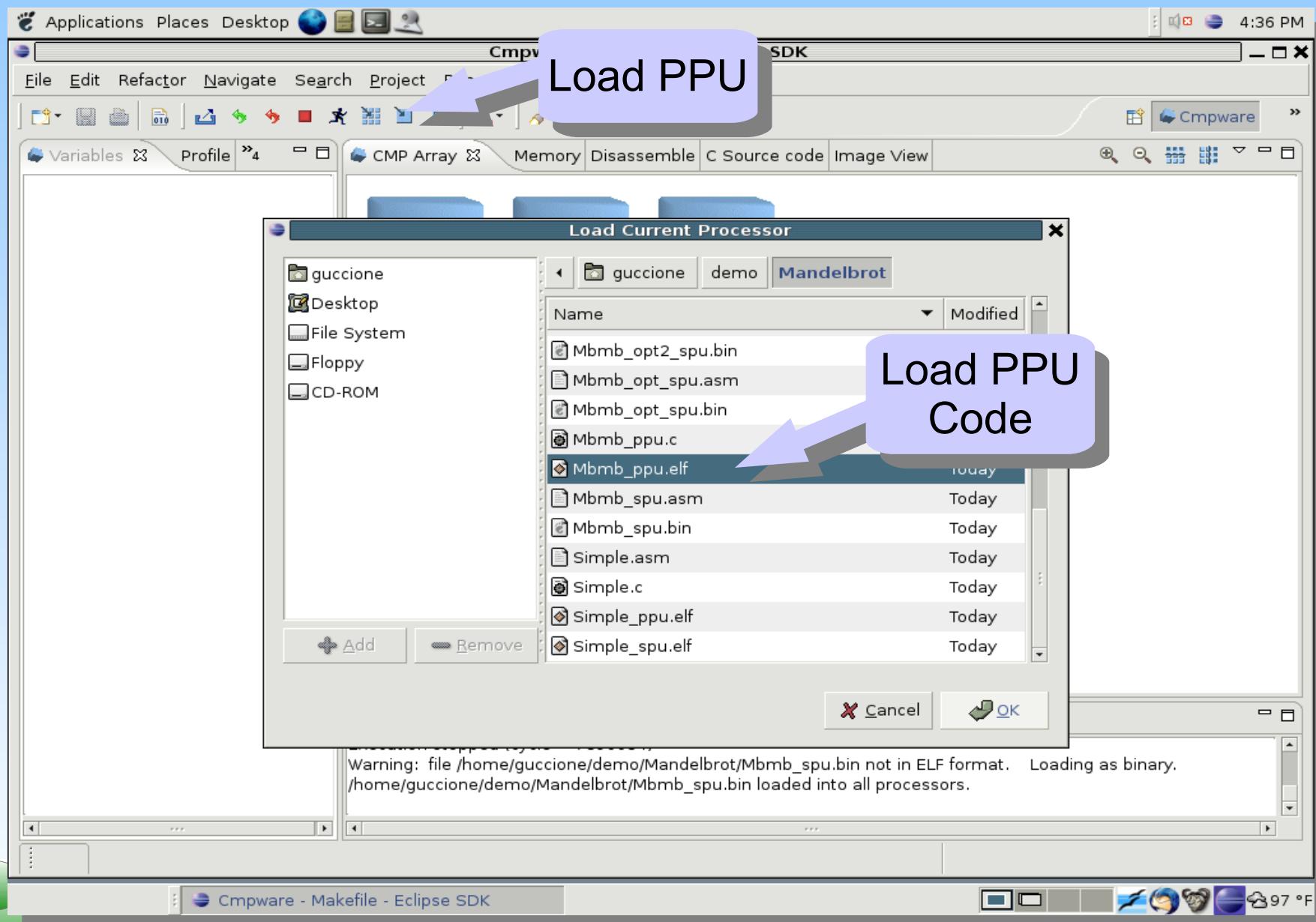


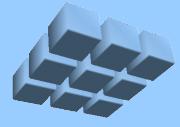
# Application III: Load SPU code



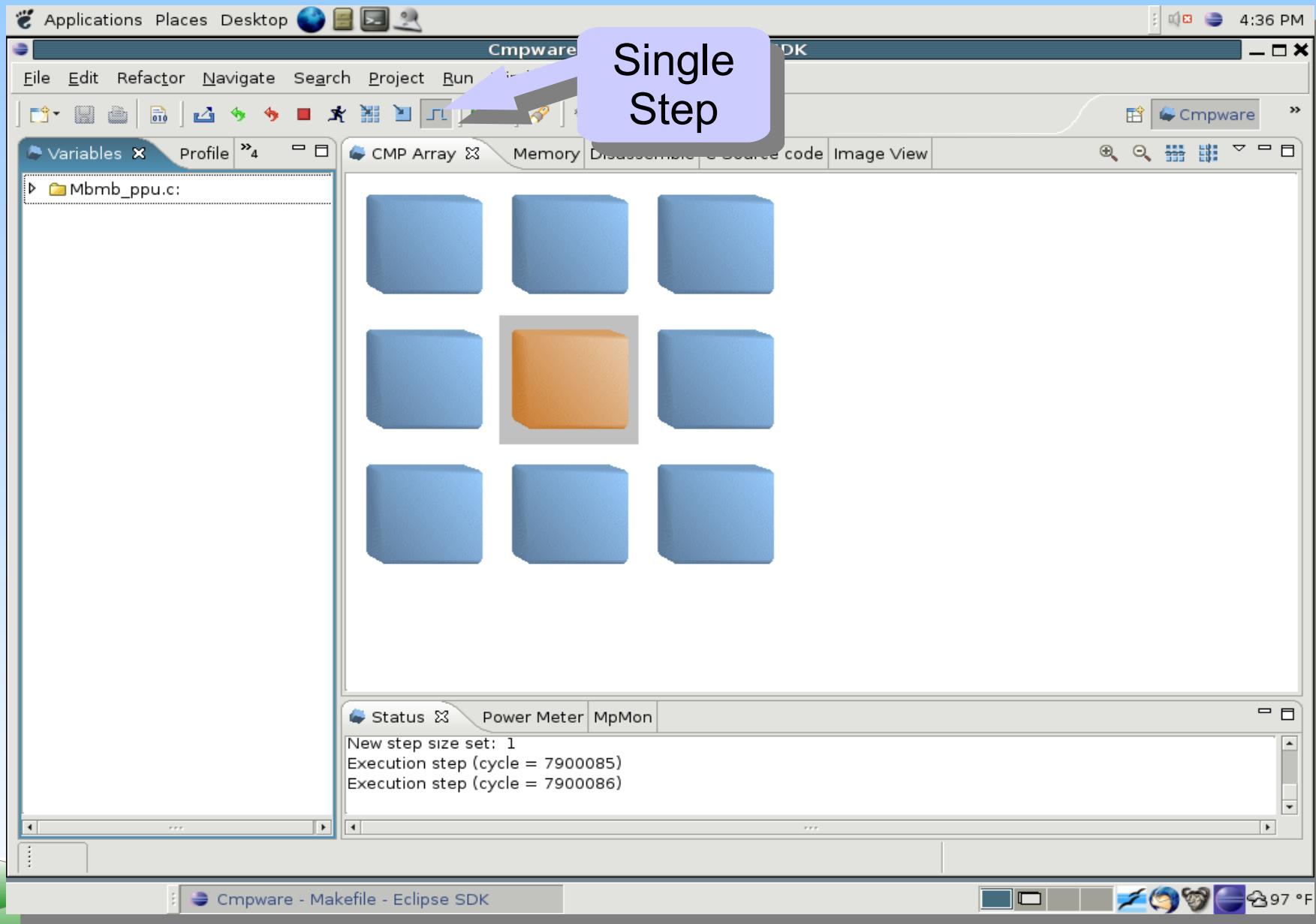


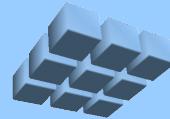
# Application III: Load PPU code



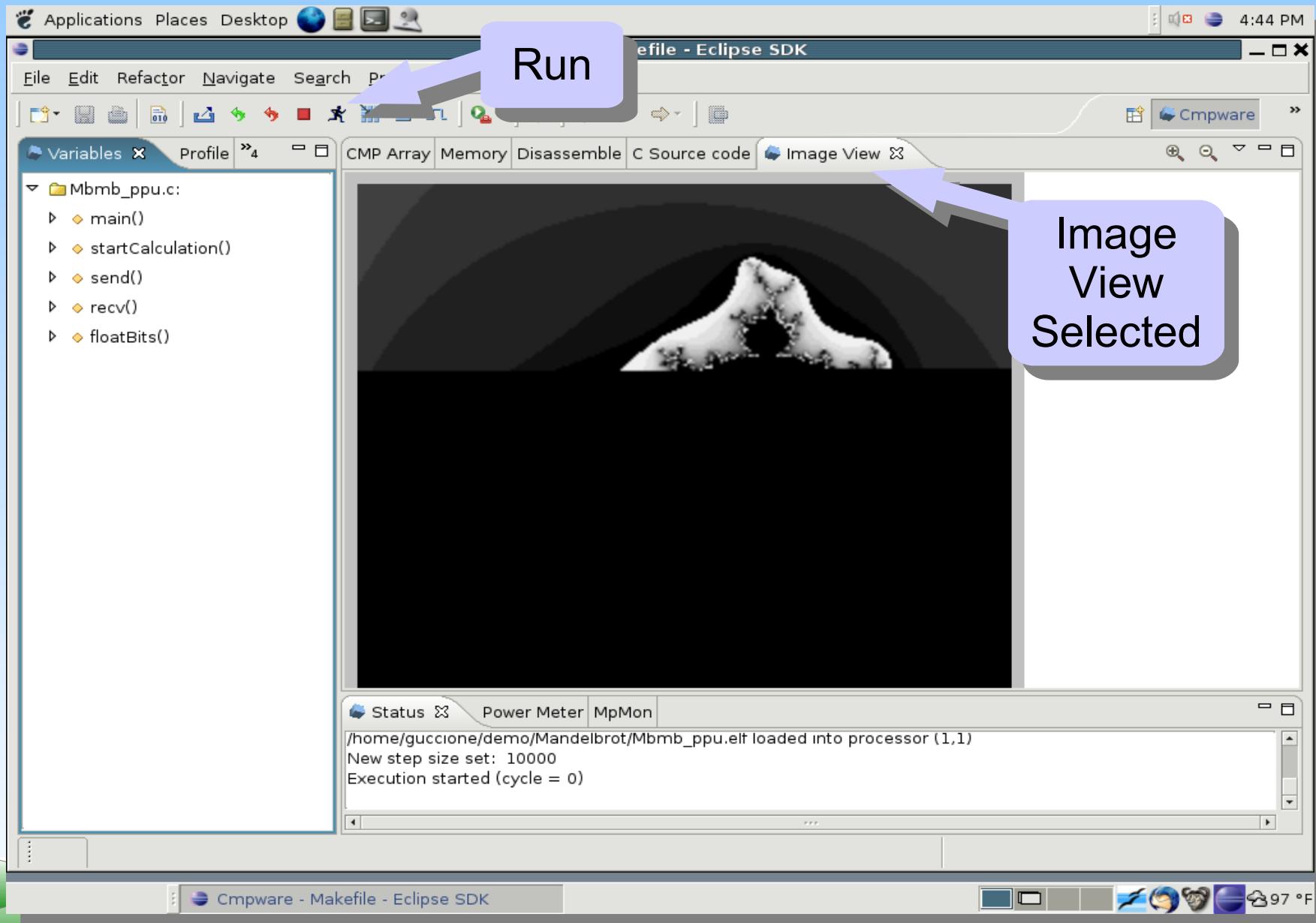


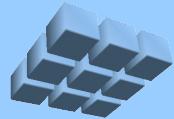
# Application III: Executing Code





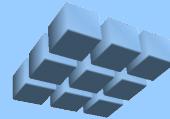
# Application III: Executing Code



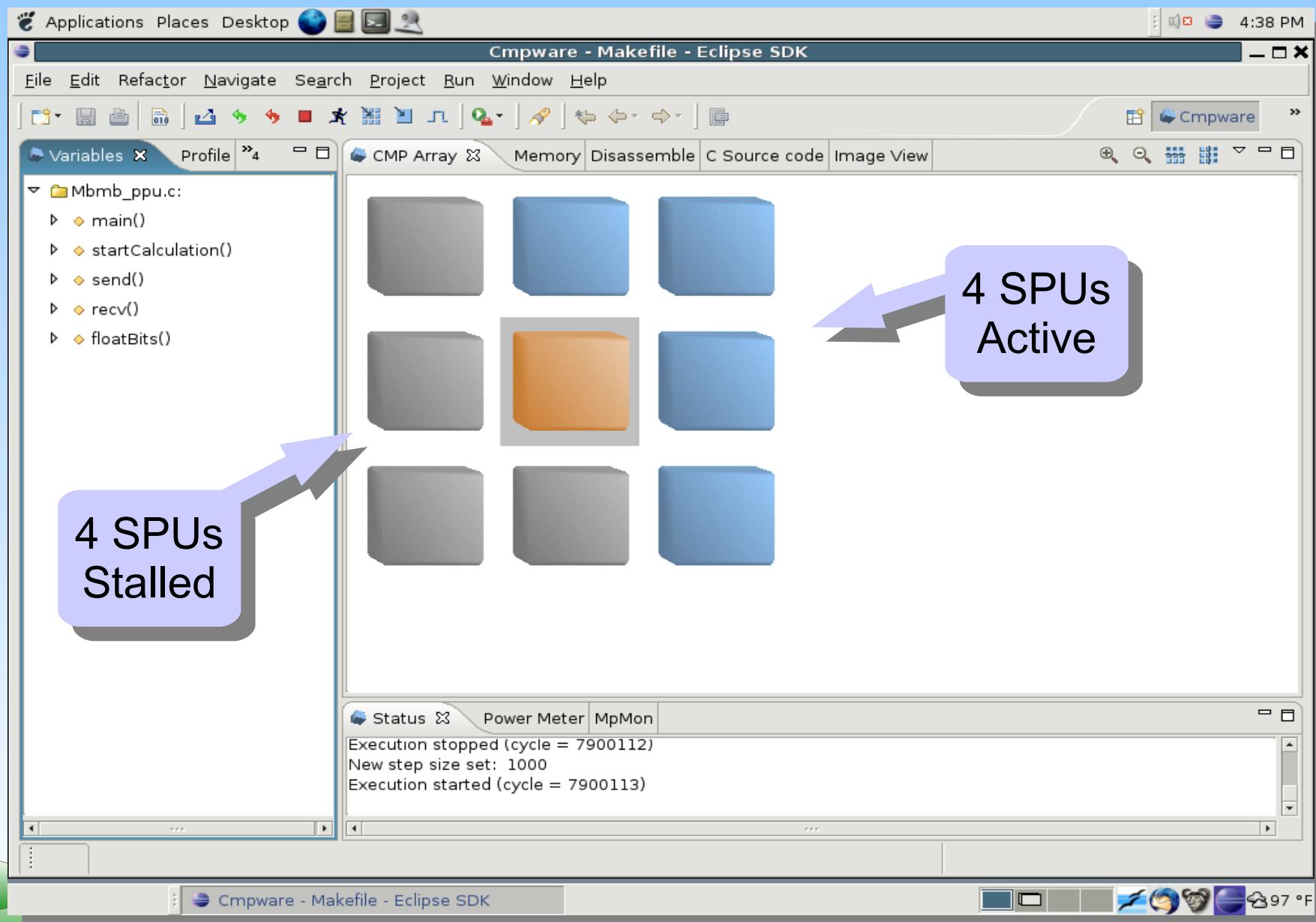


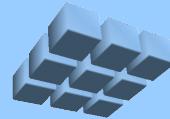
# Application III: Stalling

- SPU mailboxes:
  - **rdch** and **wrch** instructions
  - Wait for data to become available ('stall')
- Alternative to 'busy waiting' in loop
- *Cmpware* detects stalls
  - Colors processors according to activity
  - Tracks stalls in Utilization / Power Meter

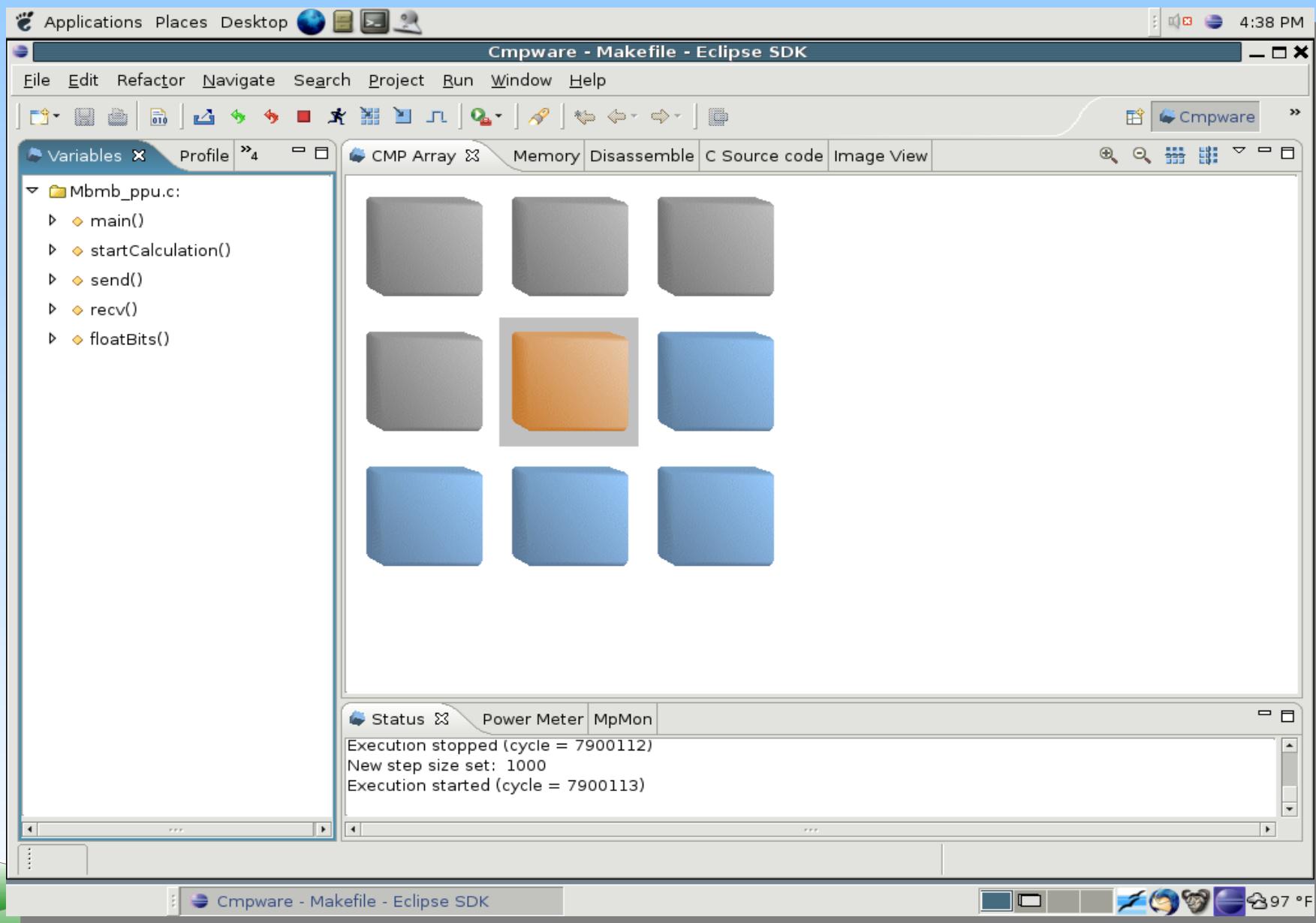


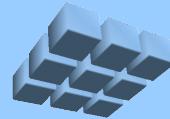
# Application III: SPU Stalls



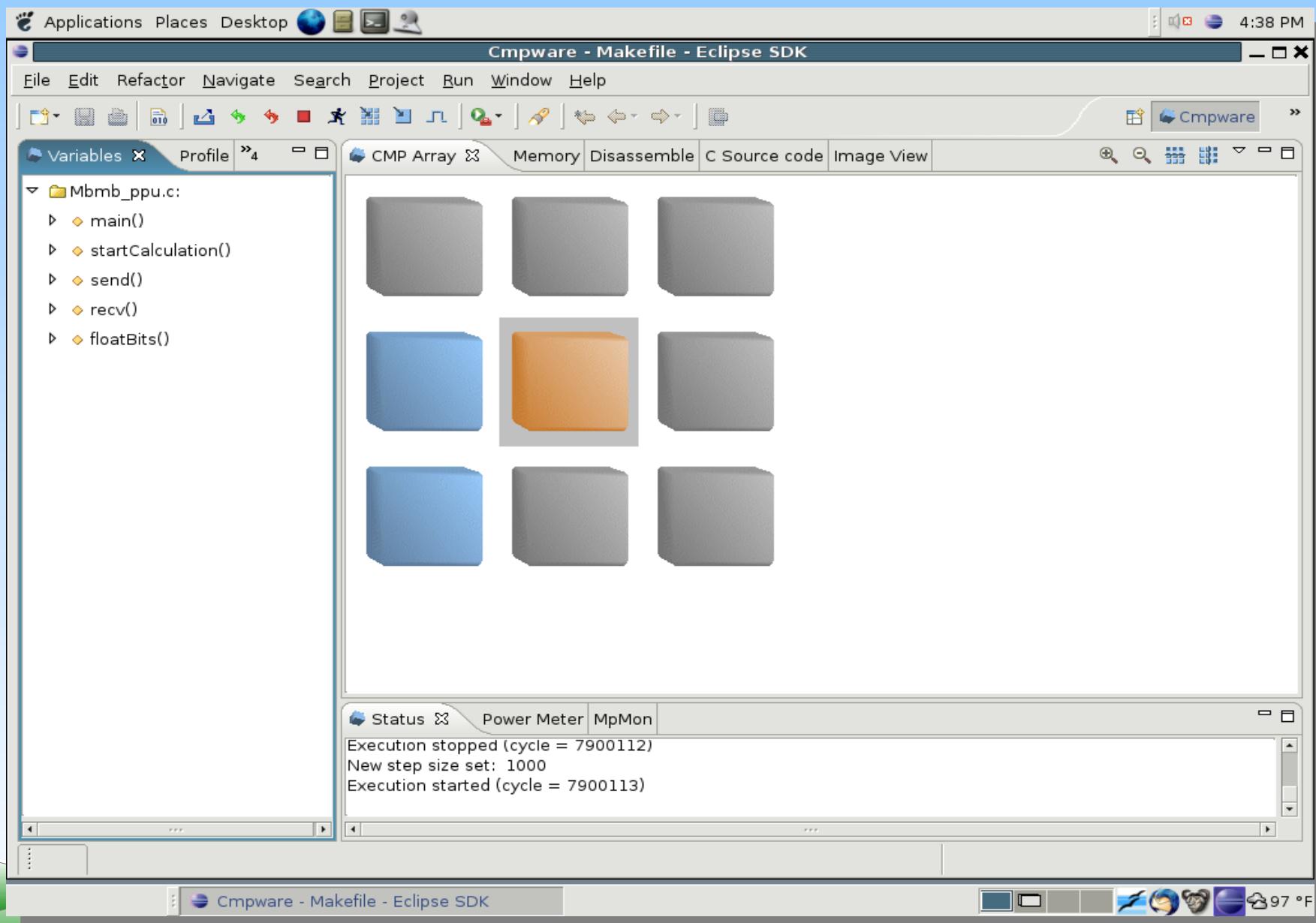


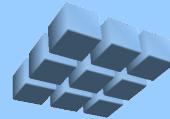
# Application III: SPU Stalls



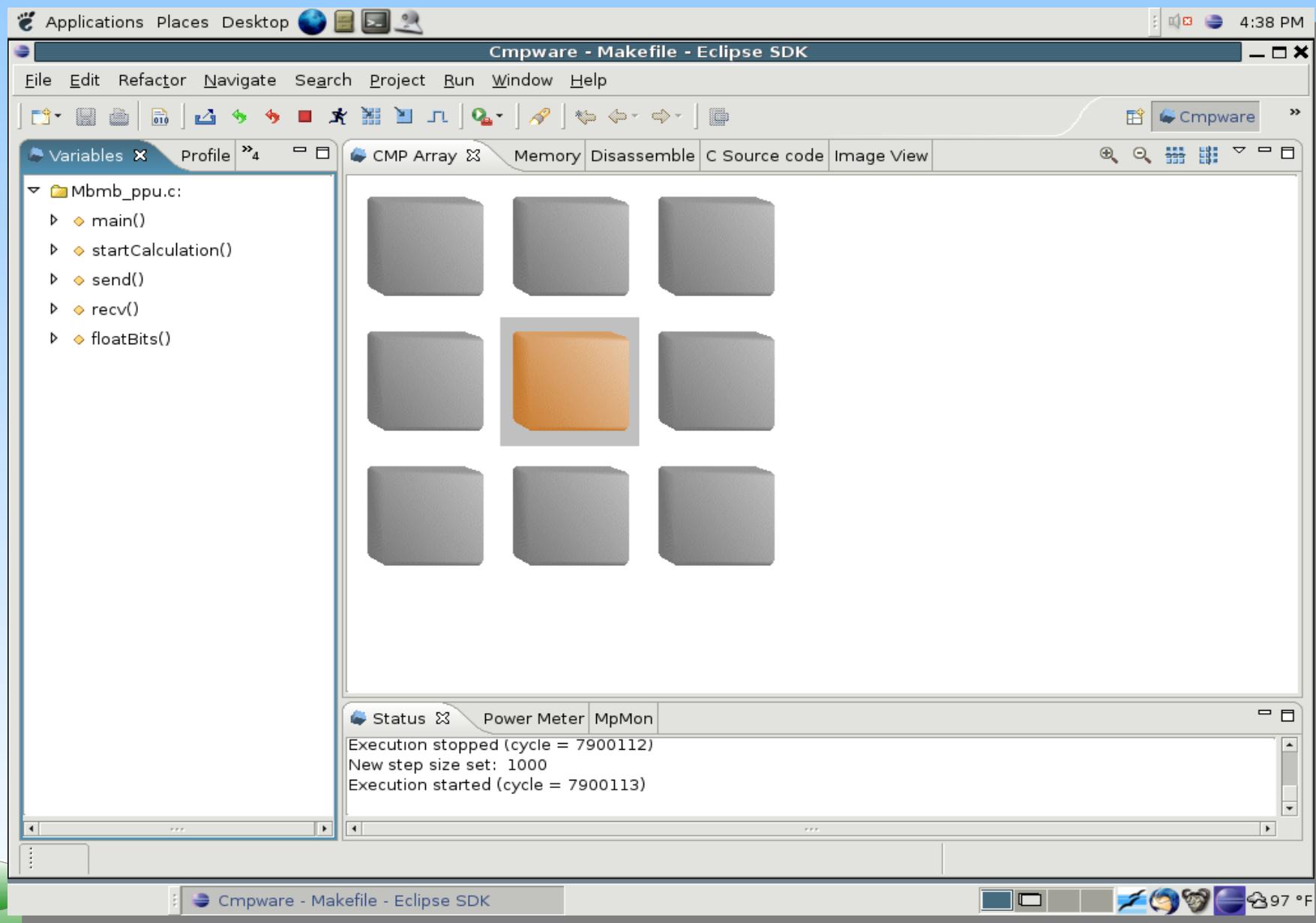


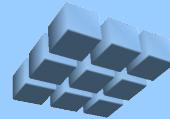
# Application III: SPU Stalls



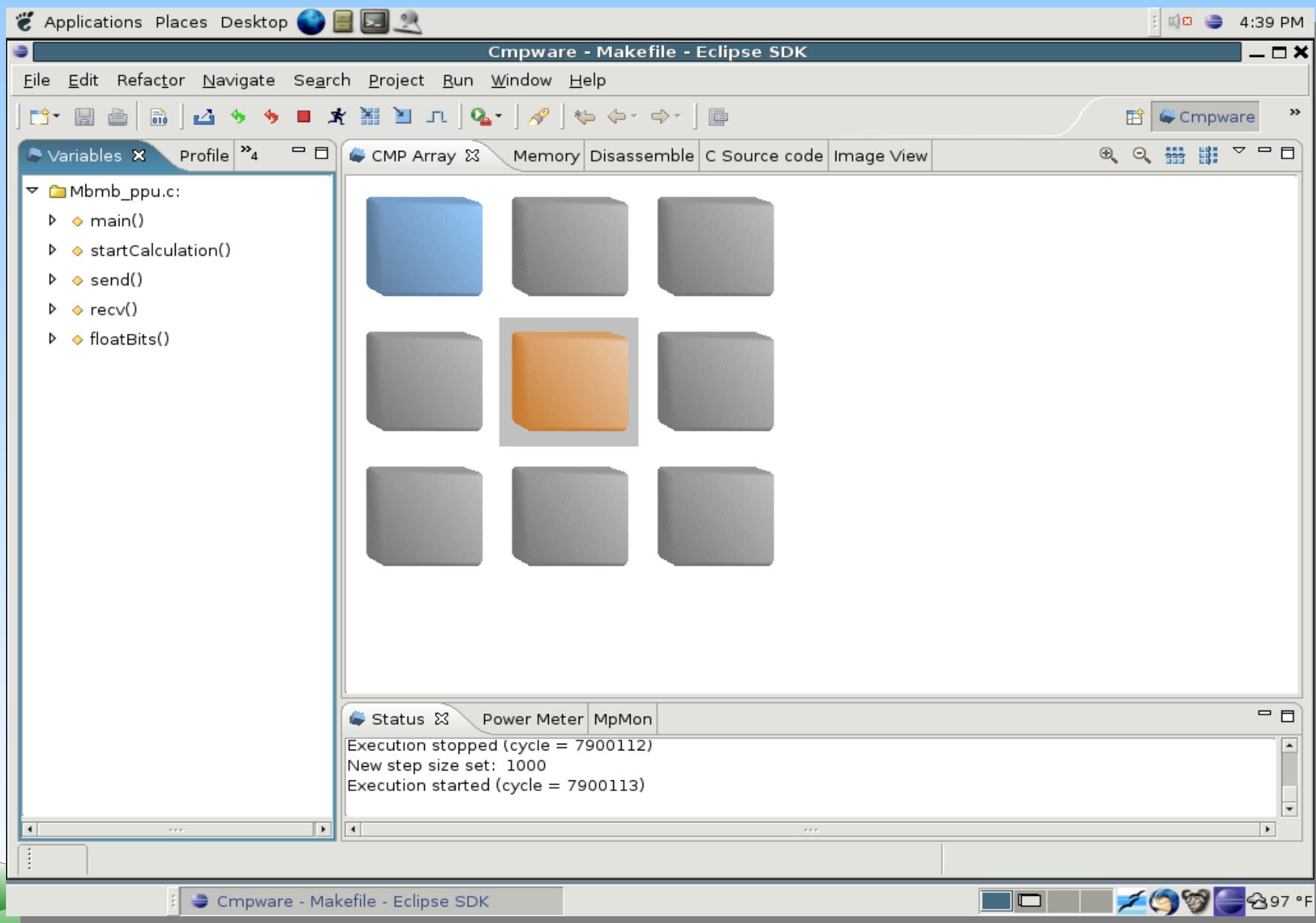


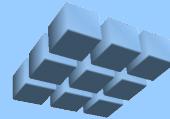
# Application III: SPU Stalls



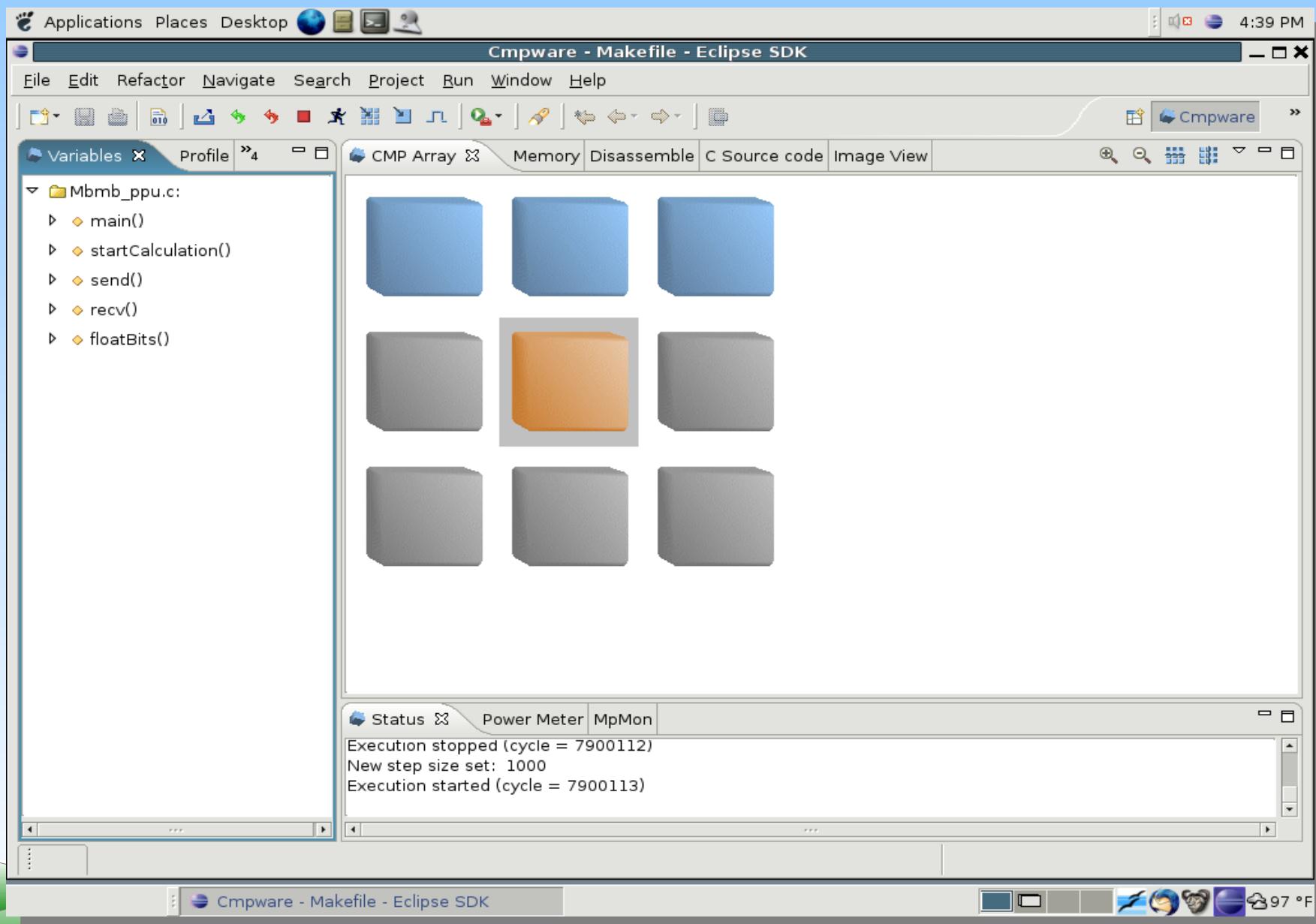


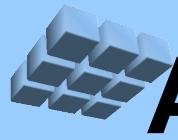
# Application III: SPU Stalls





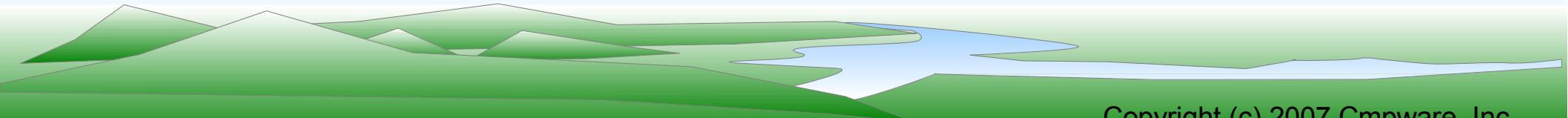
# Application III: SPU Stalls

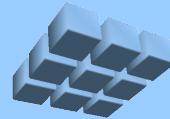




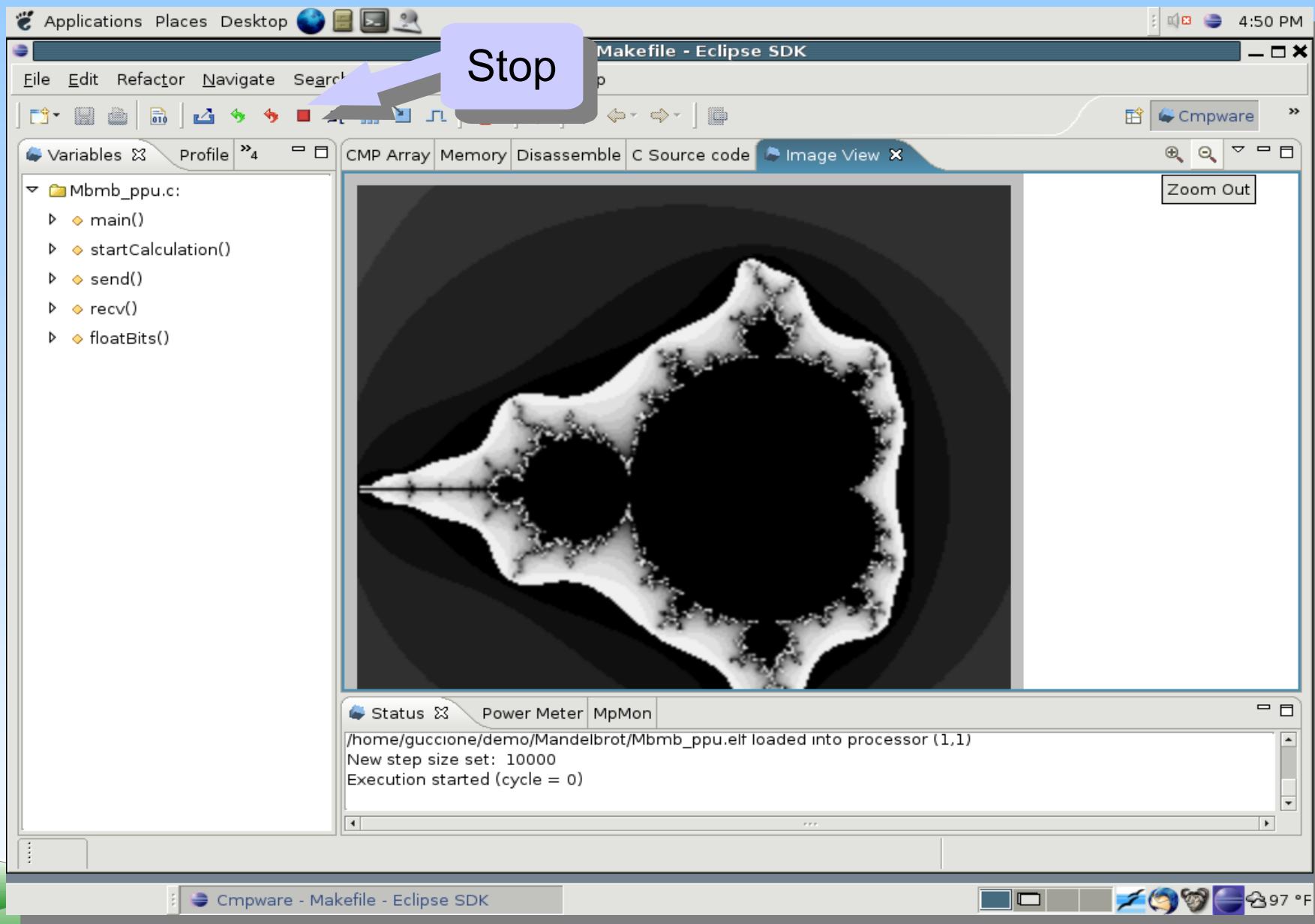
# Application III: Execution Patterns

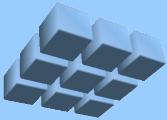
- Animated execution patterns in main display
- PPU overwhelmed 'feeding' 8 SPUs
- Approximately 30% utilization in SPUs
- Simulator defaults to one instruction per cycle
  - Verifies functional correctness
  - Gives good indication of bottlenecks
  - Identifies / animates execution patterns
  - Helps balance loads





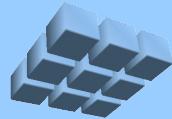
# Application III: Executing Code



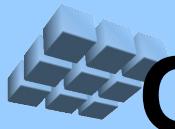


# Application III: Overview

- Mailboxes provide synchronized, blocking, serialized communication channels
- Easier to debug and analyze
- *Cmpware* tools:
  - Identify channel stalls
  - Provide animated display data
  - Track processor utilization
  - Assist in load balancing and optimization

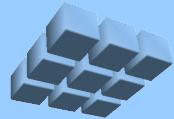


- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis
- IX. Overview



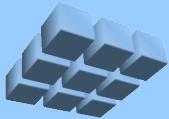
# Optimization I: SPU Dependencies

- Correctly executing hand-coded assembly is not the end of the road for SPU coding
- Performance issues depend heavily on the ordering of instructions ('scheduling')
- Results not immediately available to downstream instructions
- Instructions can wait from 2 to 6 cycles for data (longer for double precision)
- Performance can be **6x** worse than expected

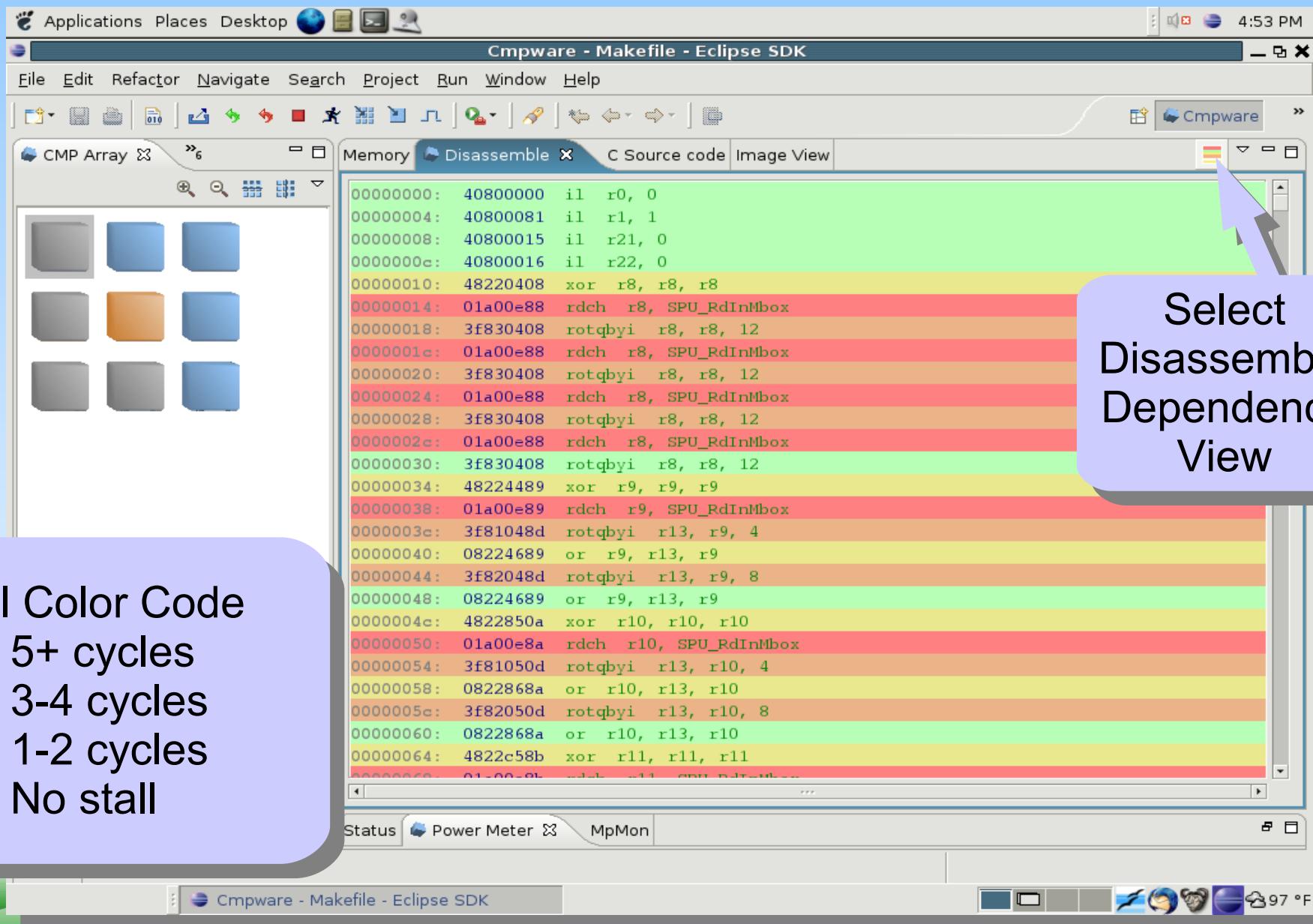


# Optimization I: Dependency View

- SPU Instruction scheduling is complex
  - Different instructions take different number of cycles – difficult to remember
  - Keeping track of register usage tedious
- *Cmpware* SPU Dependency View:
  - A quick 'snapshot' of SPU schedule
  - Intuitive color-coding of instructions
  - Interactive support for code scheduling



# Optimization I: SPU Stalls



The screenshot shows the Cmpware - Makefile - Eclipse SDK interface. The main window displays assembly code in the Disassembler view. A color-coded legend on the left maps colors to stall durations:

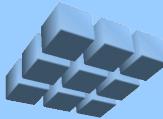
- Red: 5+ cycles
- Orange: 3-4 cycles
- Yellow: 1-2 cycles
- Green: No stall

A callout bubble points to the "Disassemble" tab in the toolbar, with the text "Select Disassemble Dependency View".

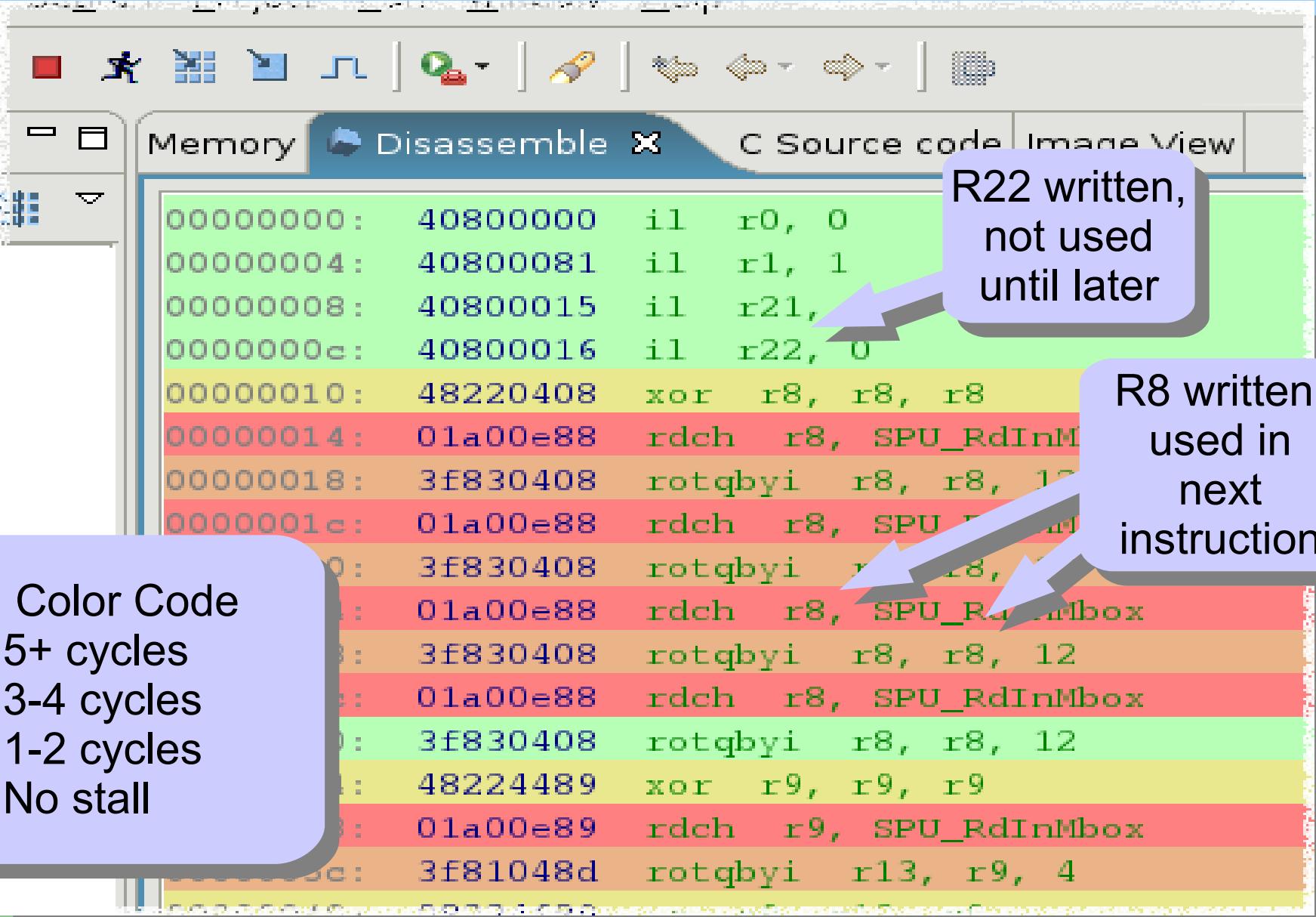
Address	OpCode	Operands
00000000	40800000	il r0, 0
00000004	40800081	il r1, 1
00000008	40800015	il r21, 0
0000000c	40800016	il r22, 0
00000010	48220408	xor r8, r8, r8
00000014	01a00e88	rdch r8, SPU_RdInMbox
00000018	3f830408	rotqbyi r8, r8, 12
0000001c	01a00e88	rdch r8, SPU_RdInMbox
00000020	3f830408	rotqbyi r8, r8, 12
00000024	01a00e88	rdch r8, SPU_RdInMbox
00000028	3f830408	rotqbyi r8, r8, 12
0000002c	01a00e88	rdch r8, SPU_RdInMbox
00000030	3f830408	rotqbyi r8, r8, 12
00000034	48224489	xor r9, r9, r9
00000038	01a00e89	rdch r9, SPU_RdInMbox
0000003c	3f81048d	rotqbyi r13, r9, 4
00000040	08224689	or r9, r13, r9
00000044	3f82048d	rotqbyi r13, r9, 8
00000048	08224689	or r9, r13, r9
0000004c	4822850a	xor r10, r10, r10
00000050	01a00e8a	rdch r10, SPU_RdInMbox
00000054	3f81050d	rotqbyi r13, r10, 4
00000058	0822868a	or r10, r13, r10
0000005c	3f82050d	rotqbyi r13, r10, 8
00000060	0822868a	or r10, r13, r10
00000064	4822c58b	xor r11, r11, r11
00000068	01a00e8b	rdch r11, SPU_RdInMbox

Stall Color Code

- 5+ cycles
- 3-4 cycles
- 1-2 cycles
- No stall



# Optimization I: SPU Stalls



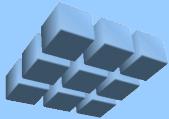
The screenshot shows a debugger interface with several tabs: Memory, Disassemble, C Source code, Image View, and others. The Disassemble tab is active, displaying assembly code. The code consists of memory addresses followed by instruction opcodes and descriptions. A color-coded legend on the left, titled 'Stall Color Code', maps colors to stall counts:

- Dark Red: 5+ cycles
- Orange: 3-4 cycles
- Yellow: 1-2 cycles
- Green: No stall

Annotations with arrows point to specific instructions:

- An arrow points to the instruction at address 0x000000c: 40800016 il r22, 0, with the text "R22 written, not used until later".
- An arrow points to the instruction at address 0x0000010: 48220408 xor r8, r8, r8, with the text "R8 written, used in next instruction".

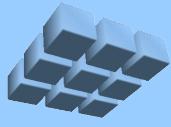
Address	Opcode	Description	Color
00000000	40800000	il r0, 0	Green
00000004	40800081	il r1, 1	Green
00000008	40800015	il r21,	Green
0000000c	40800016	il r22, 0	Green
00000010	48220408	xor r8, r8, r8	Yellow
00000014	01a00e88	rdch r8, SPU_RdInM	Red
00000018	3f830408	rotqbyi r8, r8, 12	Red
0000001c	01a00e88	rdch r8, SPU_RdInM	Red
00000020	3f830408	rotqbyi r8, r8, 12	Red
00000024	01a00e88	rdch r8, SPU_RdInM	Red
00000028	3f830408	rotqbyi r8, r8, 12	Red
0000002c	01a00e88	rdch r8, SPU_RdInM	Red
00000030	3f830408	rotqbyi r8, r8, 12	Red
00000034	48224489	xor r9, r9, r9	Yellow
00000038	01a00e89	rdch r9, SPU_RdInM	Red
0000003c	3f81048d	rotqbyi r13, r9, 4	Red



# Optimization I: Statistics

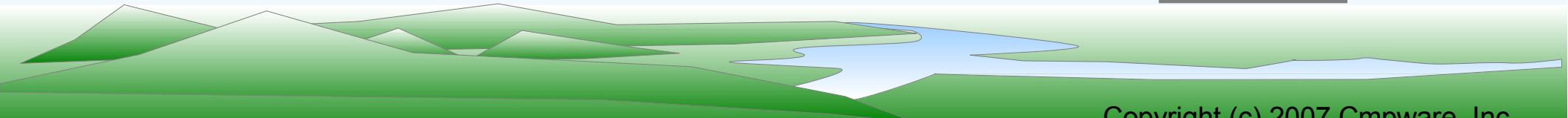
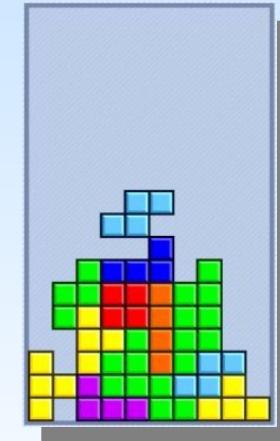
- 60 instructions, 167 - 190+ cycles
- **2x - 3x** penalty for dependency stalls

<u>Instructions</u>	<u>Stalls</u>	<u>Stall Cycles</u>
25	No stalls (green)	0
11	1-2 stalls (yellow)	11 – 22
12	3-4 stalls (orange)	36 – 48
12	5+ stalls (red)	60
<b>Total: 60</b>		<b>107 – 130+</b>

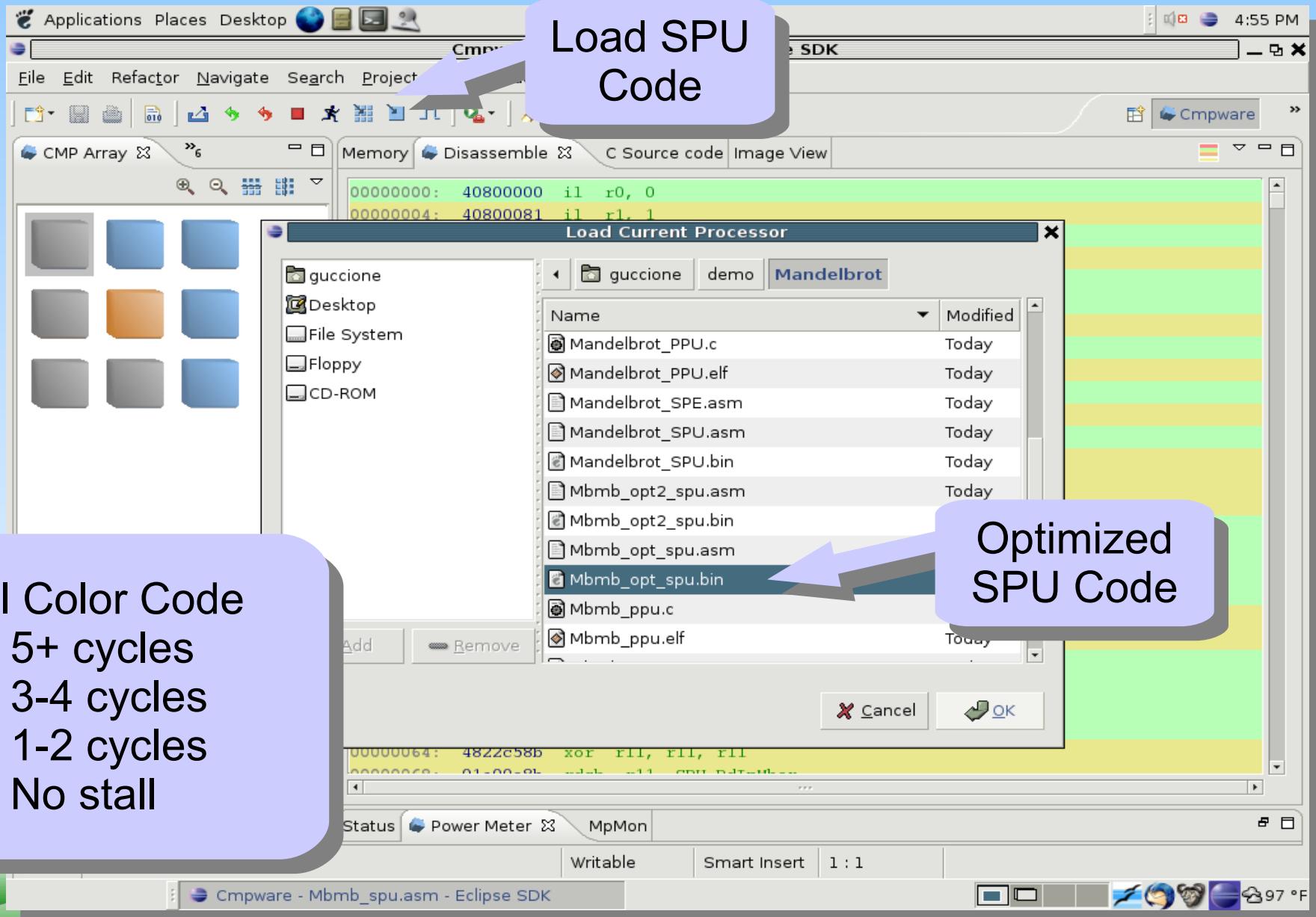


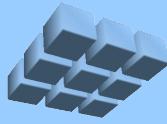
# Optimization I: Optimization Tips

- Load data early
- Use lots of registers – register reuse can make scheduling difficult
- Do multiple interleaved calculations at a time
- 'Abstract' away from the algorithm – just think registers and cycles
- It's sort of like 'Tetris' :^)



# Optimization I: Optimized Code





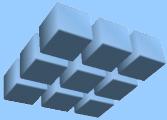
# Application III: Optimized Code

Stall Color Code

- █ 5+ cycles
- █ 3-4 cycles
- █ 1-2 cycles
- █ No stall

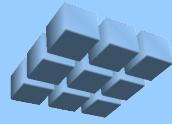
All Stalls Removed

```
000000a0: 08238204 or r4, r4, r14
000000a4: 08244285 or r5, r5, r17
000000a8: 40800012 il r18, 0
000000ac: 40800013 il r19, 0
000000b0: 58c48919 fm r25, r18, r18
000000b4: 58c4c99a fm r26, r19, r19
000000b8: 58c4c91c fm r28, r18, r19
000000bc: 58a68c92 fs r18, r25, r26
000000c0: 58870e13 fa r19, r28, r28
000000c4: 58808912 fa r18, r18, r2
000000c8: 5880c993 fa r19, r19, r3
000000cc: 58c4891a fm r26, r18, r18
000000d0: 58c4c99b fm r27, r19, r19
000000d4: 40200000 nop
000000d8: 58868d9c fa r28, r27, r26
000000dc: 40200000 nop
000000e0: 58410e1d fcgt r29, r28, r4
000000e4: 40200000 nop
000000e8: 08274b16 or r22, r22, r29
000000ec: 40200000 nop
000000f0: 18204b1e and r30, r22, r1
000000f4: 1cfffc285 ai r5, r5, -1
000000f8: 18078b97 a r23, r23, r30
000000fc: d4fff685 brnz r5, -19
00000100: 3fe10b98 shlqbyi r24, r23, 4
00000104: 3fe20b99 shlqbyi r25, r23, 8
00000108: 3fe30b9a shlqbyi r26, r23, 12
```

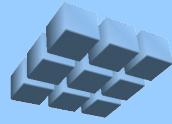


# Optimization I: Overview

- SPU performance highly dependent on dependencies ('schedule')
- An unusual idea for many programmers
- *Cmpware* SPU Dependency View:
  - Give a 'snapshot' of dependencies
  - Fast and interactive
  - Permits lots of experimentation
- Can dramatically improve SPU performance
- **2x - 3x** speedup in Mandelbrot SPU code

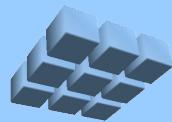


- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis
- IX. Overview



# Optimization II: SPU Dual Issue

- SPU can execute two instructions in a cycle
- Two pipelines not identical
  - **'Pipe 0' instructions:** arithmetic, logical, others
  - **'Pipe 1' instructions:** branches, loads, stores, others
- Pipe 0 instructions should be aligned on 'even' addresses (ending in 0x0 or 0x8)
- Pipe 1 instructions should be aligned on 'odd' addresses (ending in 0x4 or 0xc)



# Optimization II: Alignment View

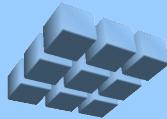
The screenshot shows the Eclipse SDK interface with the project "Cmpware - Mbmb\_spu.asm". The "Disassemble" tab is selected, displaying assembly code. The code is color-coded by alignment: yellow for incorrectly aligned instructions and green for correctly aligned ones. A callout bubble on the left explains the color coding. Another callout bubble on the right points to the "Display performance data" button.

Alignment Color Code:

- Yellow: Incorrectly aligned
- Green: Correctly aligned

Select Instruction Alignment View

Address	Instruction	Color
00000068:	3f83020e rotqbyi r14, r4, r2	Green
0000006c:	3f81028f rotqbyi r15, r5, 4	Yellow
00000070:	3f820290 rotqbyi r16, r5, 8	Yellow
00000074:	3f830291 rotqbyi r17, r5, 12	Yellow
00000078:	08218102 or r2, r2, r6	Green
0000007c:	08224183 or r3, r3, r9	Yellow
00000080:	08230204 or r4, r4, r12	Yellow
00000084:	0823c285 or r5, r5, r15	Yellow
00000088:	0821c102 or r2, r2, r7	Yellow
0000008c:	08228183 or r3, r3, r10	Yellow
00000090:	08234204 or r4, r4, r13	Yellow
00000094:	08240285 or r5, r5, r16	Yellow
00000098:	08220102 or r2, r2, r8	Green
0000009c:	0822c183 or r3, r3, r11	Yellow
000000a0:	08238204 or r4, r4, r14	Yellow
000000a4:	08244285 or r5, r5, r17	Yellow
000000a8:	40800012 il r18, 0	Green
000000ac:	40800013 il r19, 0	Yellow
000000b0:	58c48919 fm r25, r18, r18	Green
0b4:	58c4c99a fm r26, r19, r19	Yellow
0b8:	58c4c91c fm r28, r18, r19	Yellow
0bc:	58a68c92 fs r18, r25, r26	Yellow
0c0:	58870e13 fa r19, r28, r28	Green
0c4:	58808912 fa r18, r18, r2	Yellow
0c8:	5880c993 fa r19, r19, r3	Green
0cc:	58c4891a fm r26, r18, r18	Yellow
0d0:	58c4c99b fm r27, r19, r19	Yellow

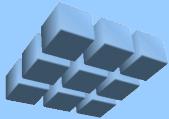


# Optimization II: Alignment

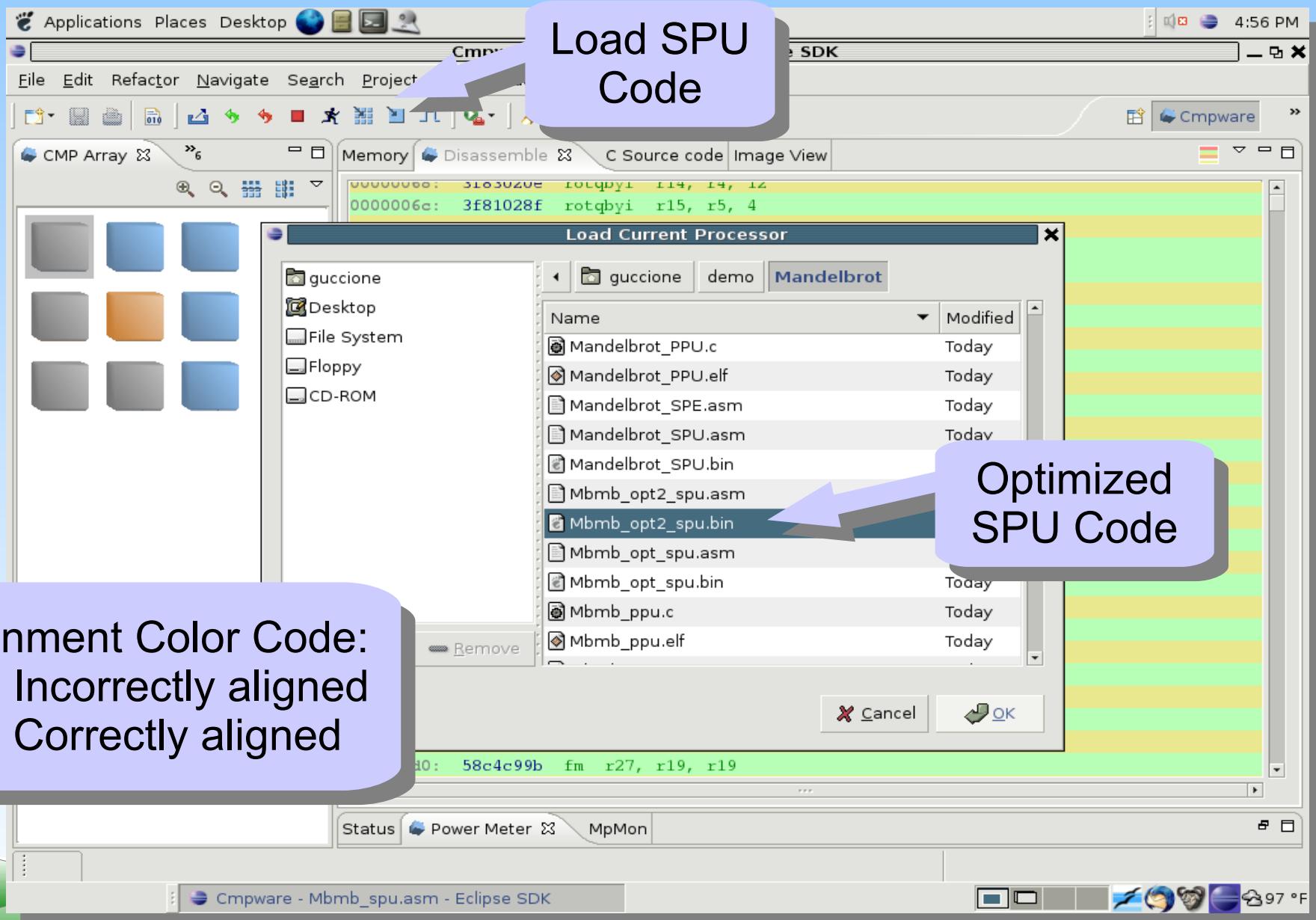
- Worst case:
  - One instruction per cycle
  - yellow / green 'stripes'
  - 50% SPU utilization
- Best case:
  - All green
  - Completely aligned
  - 100% SPU utilization (2x speedup)
- All yellow: only one 'nop' away from 100%

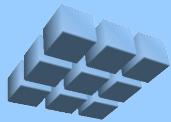
The screenshot shows a debugger interface with several tabs at the top: Memory, Disassemble, C Source, etc. The Disassemble tab is active, displaying assembly code. The code consists of 16 lines, each starting with a memory address (e.g., 00000060, 0000006c, ..., 0000009c) followed by an instruction and its operands. The assembly instructions are mostly 'rotqbyi' and 'or' with various registers (r14, r15, r16, r17, r2, r3, r4, r5, r6, r7, r8). The background of the code area has horizontal stripes of yellow and green, indicating memory alignment. The yellow stripes are located at addresses 00000060, 0000006c, 00000070, 00000074, 00000078, 0000007c, 00000080, 00000084, 00000088, 0000008c, 00000090, 00000094, 00000098, and 0000009c. The green stripes are located at addresses 00000064, 00000068, 00000072, 00000076, 00000082, 00000086, 00000092, and 00000096.

Address	Instruction	Operands
00000060	3103020E	rotqbyi r14,
00000064	3F81028F	rotqbyi r15,
00000068	3F820290	rotqbyi r16,
00000072	3F830291	rotqbyi r17,
00000076	08218102	or r2, r2, r6
0000007C	08224183	or r3, r3, r7
00000080	08230204	or r4, r4, r8
00000084	0823C285	or r5, r5, r9
00000088	0821C102	or r2, r2, r10
0000008C	08228183	or r3, r3, r11
00000090	08234204	or r4, r4, r12
00000094	08240285	or r5, r5, r13
00000098	08220102	or r2, r2, r14
0000009C	0822C183	or r3, r3, r15



# Optimization II: Aligned Code





# Optimization II: Aligned Code

Applications Places Desktop 4:57 PM

Cmpware - Mbmb\_spu.asm - Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

CMP Array

Memory Disassemble C Source code Image View

000000d4: 00200000 lnop  
000000d8: 58c4c91c fm r28, r18, r19  
000000dc: 00200000 lnop  
000000e0: 58a68c92 fs r18, r25, r26  
000000e4: 00200000 lnop  
000000e8: 58870e13 fa r19, r28, r28  
000000ec: 00200000 lnop  
000000f0: 58808912 fa r18, r18, r2  
000000f4: 00200000 lnop  
000000f8: 5880c993 fa r19, r19, r3  
000000fc: 00200000 lnop  
00000100: 58c4891a fm r26, r18, r18  
00000104: 00200000 lnop  
00000108: 58c4c99b fm r27, r19, r19  
0000010c: 00200000 lnop  
00000110: 58868d9c fa r28, r27, r26  
00000114: 00200000 lnop  
00000118: 58410e1d fcgt r29, r28, r4  
0000011c: 00200000 lnop  
00000120: 08274b16 or r22, r22, r29  
00000124: 00200000 lnop  
00000128: 18204b1e and r30, r22, r1  
0000012c: 00200000 lnop  
00000130: 1cffc285 ai r5, r5, -1  
00000134: 00200000 lnop  
00000138: 18078b97 a r23, r23, r30  
0000013c: d4ffff605 breq r5, -20

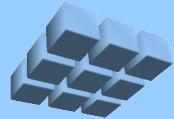
Completely Aligned Code (note nops / lnops)

Alignment Color Code:  
Incorrectly aligned (Yellow)  
Correctly aligned (Green)

Status Power Meter MpMon

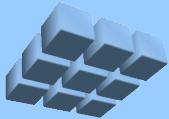
Cmpware - Mbmb\_spu.asm - Eclipse SDK

97 °F



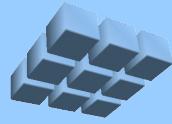
# Optimization II: Alignment Tips

- Start from the top and work down
- Move instructions upward
- Don't break existing dependencies!
- Insert **nop/lnop** as a last resort
- **Nop/lnops :**
  - Increase code size
  - But may reduce overall cycle count
  - Nice to have **nops** in code as 'documentation'

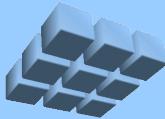


# Optimization II: Overview

- SPU performance highly dependent on instruction alignment
- Another unusual idea for many programmers
- *Cmpware* SPU Alignment View:
  - Give a 'snapshot' of instruction alignment
  - Fast and interactive
  - Permits lots of experimentation
- Can quickly improve SPU performance
- Nearly **2x** speedup in Mandelbrot SPU code



- I. Introduction
- II. Installing the *Cmpware CMP-DK*
- III. Application I: A Simple Program
- IV. Application II: Shared Memory
- V. Application III: Mailboxes
- VI. Optimization I: SPU Dependencies
- VII. Optimization II: SPU Dual Issue
- VIII. System Level Analysis**
- IX. Overview

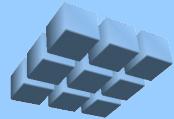


# System Level Analysis: Optimization Statistics

- Each optimization phase adds instructions ...  
... but reduces cycle count
- Approx. **4x** performance increase

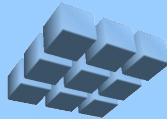
	<u>Instructions</u>	<u>Stalls</u>	<u>I.P.C.*</u>	<u>Cycles</u>
Naïve	60	107 – 130+	1	167 – 190+
Opt. I	73	0	1	73
Opt. II	98	0	2	49

\* I.P.C. is 'Instructions Per Clock' for SPU dual issue pipeline.



# System Level Analysis

- Use various *Cmpware* tools together:
  - Processor Utilization indicates which nodes to optimize
  - Profiling indicates areas on nodes to optimize
  - Stall data / profiling aids in partitioning
- Customize the IDE:
  - Any view can be accessed in <2 mouse clicks
  - Moving / resizing windows streamlines development

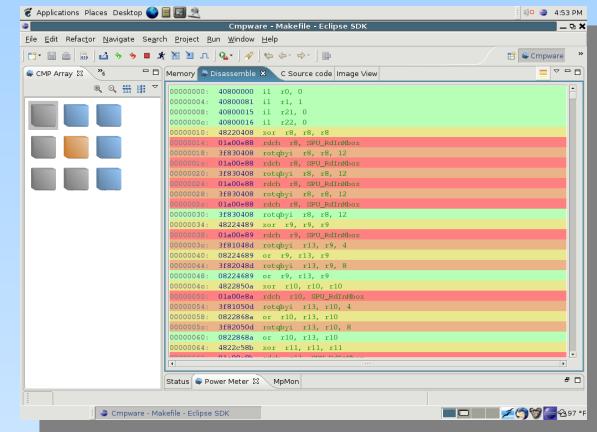


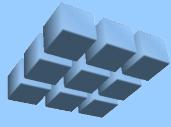
# Putting It All Together

- In the *Cmpware CMP-DK* for the Cell BE:

1. Edit ...
2. Compile ...
3. Execute ...
4. and Debug multicore code
5. Find bottlenecks and 'hot spots'
6. Optimize SPU dependencies
7. Optimize SPU instruction alignment

*... all in one Eclipse-based IDE*



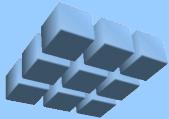


# Cmpware CMP-DK for the Cell BE

- Eclipse / Java based
- Runs 'everywhere'
- Completely self-contained
- Compact: 1.5 MB 'plugin'
- Easy to install (seconds)



- ***Increase programmer productivity***
- ***Increase Cell BE software performance***



# Resources

- Cmpware, Inc: <http://www.cmpware.com/>
- Cmpware Cell BE Update Site:  
<http://www.cmpware.com/cellbe/>
- All code from this presentation available at:  
[http://www.cmpware.com/cellbe/CmpwareCellBE\\_demo.zip](http://www.cmpware.com/cellbe/CmpwareCellBE_demo.zip)
- Cmpware License Request: [info@cmpware.com](mailto:info@cmpware.com)
- IBM Cell tools for Windows / Cygwin  
<http://cellbe-cygwin.cvs.sourceforge.net/cellbe-cygwin/cellbe-cygwin/>