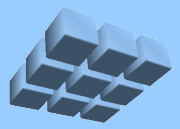
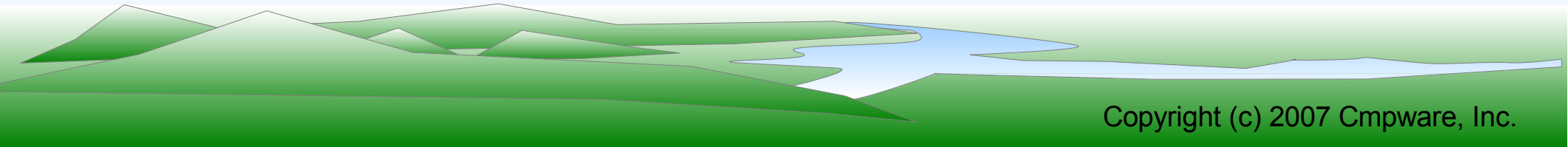
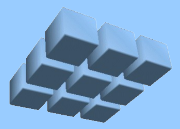


Modeling and Programming the Cell BE using the Cmpware CMP-DK

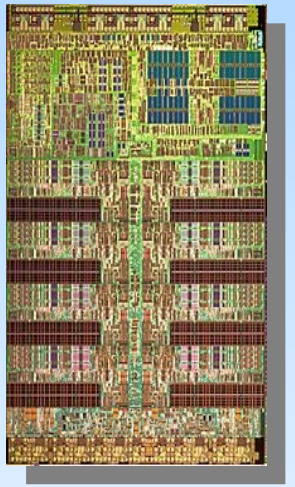
Steven A. Guccione
Cmpware, Inc.



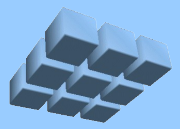


Multicore Processing

- Multicore devices increasingly used for high performance computation
- All modern CPUs are multicore
- Multicore offers:
  - High performance
  - Low power
  - Simplified hardware design

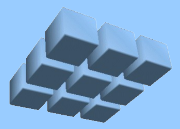


... but is more difficult to program

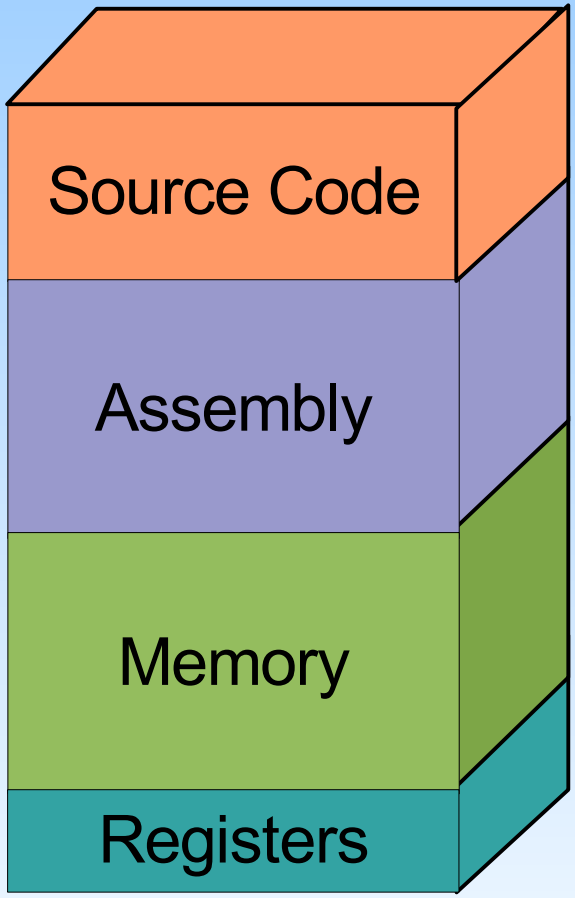


The Multicore Software Problem

- ***N*** cores produce ***N*** times the data
- Processors states constantly changing
- Multicore development and debug becomes a complex exercise in *managing data*
- *Cmpware CMP-DK* provides fast and easy access to **all** multicore state data
- The simulation-based approach in the *Cmpware CMP-DK* gives superior control and access to the multicore architecture

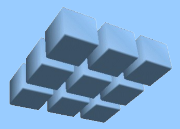


The Processor State Data Space

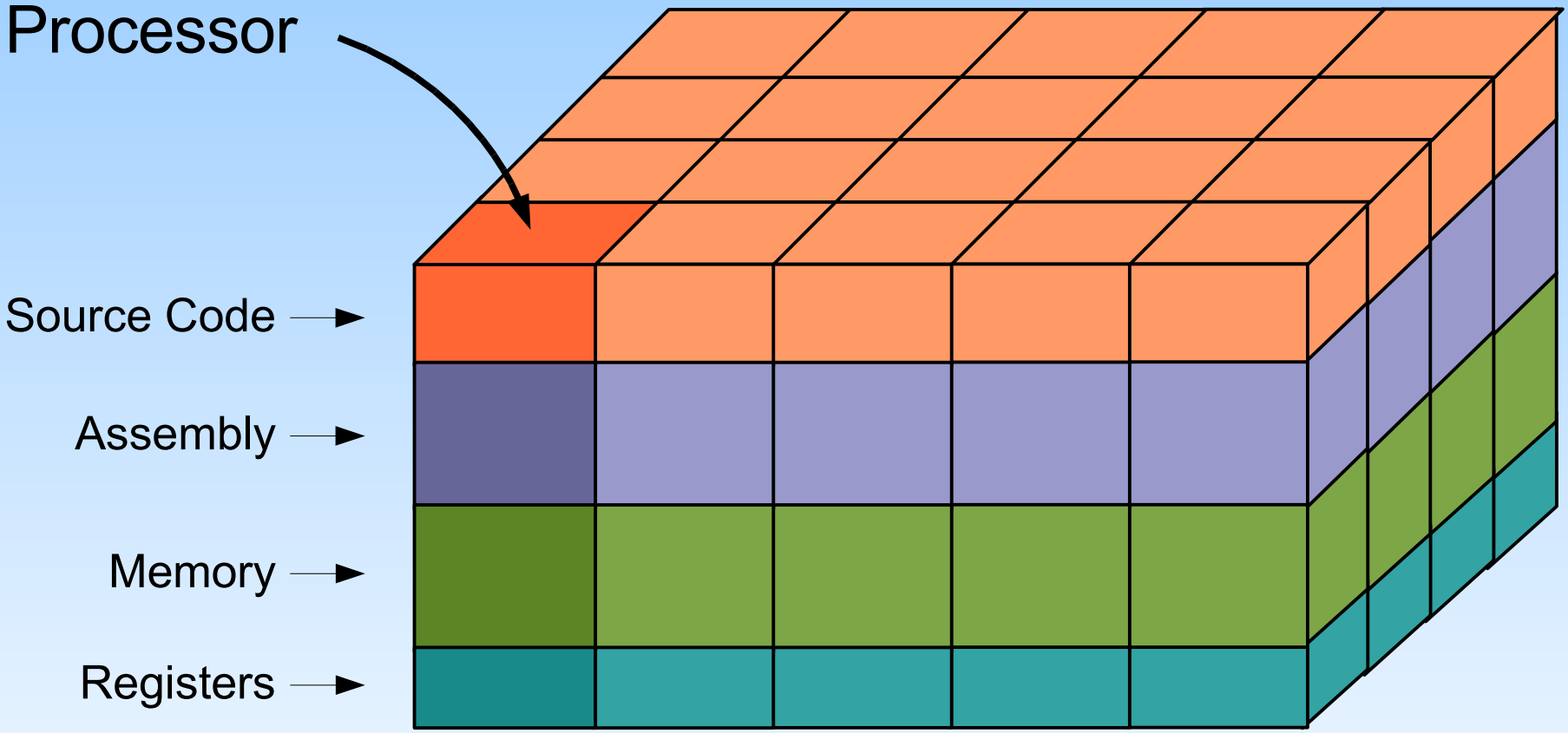


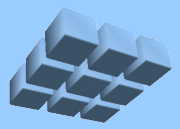
High Level Data

Low Level Data



The Multicore Data Space



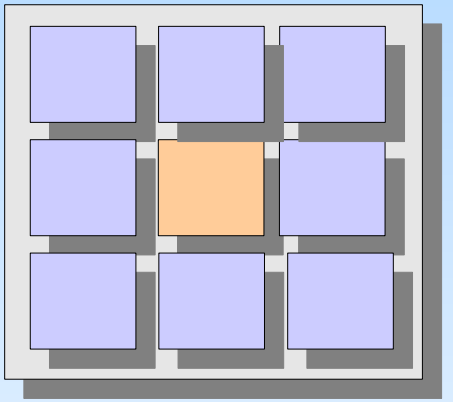


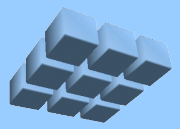
The *Cmpware CMP-DK*

- A multicore architecture modeling and software development environment
- A *'programmer's view'* of the hardware
- The *Cmpware CMP-DK* is used to:
  1. Model a multicore architectures
  2. Write software for this architecture
  3. Execute compiled code on the models
  4. View the results interactively in the IDE

The Cell BE Simulation Model

- Cmpware *Cell BE* Simulation model:
  - ***PowerPC* core**: 657 lines of code (';')
  - ***PowerPC* FP extensions**: 287 lines
  - **SPE core**: 924 lines
  - **System code**: 48 lines
- Supplies all IDE display data
- Built-in assemblers and disassemblers
- **4M+** operations per second

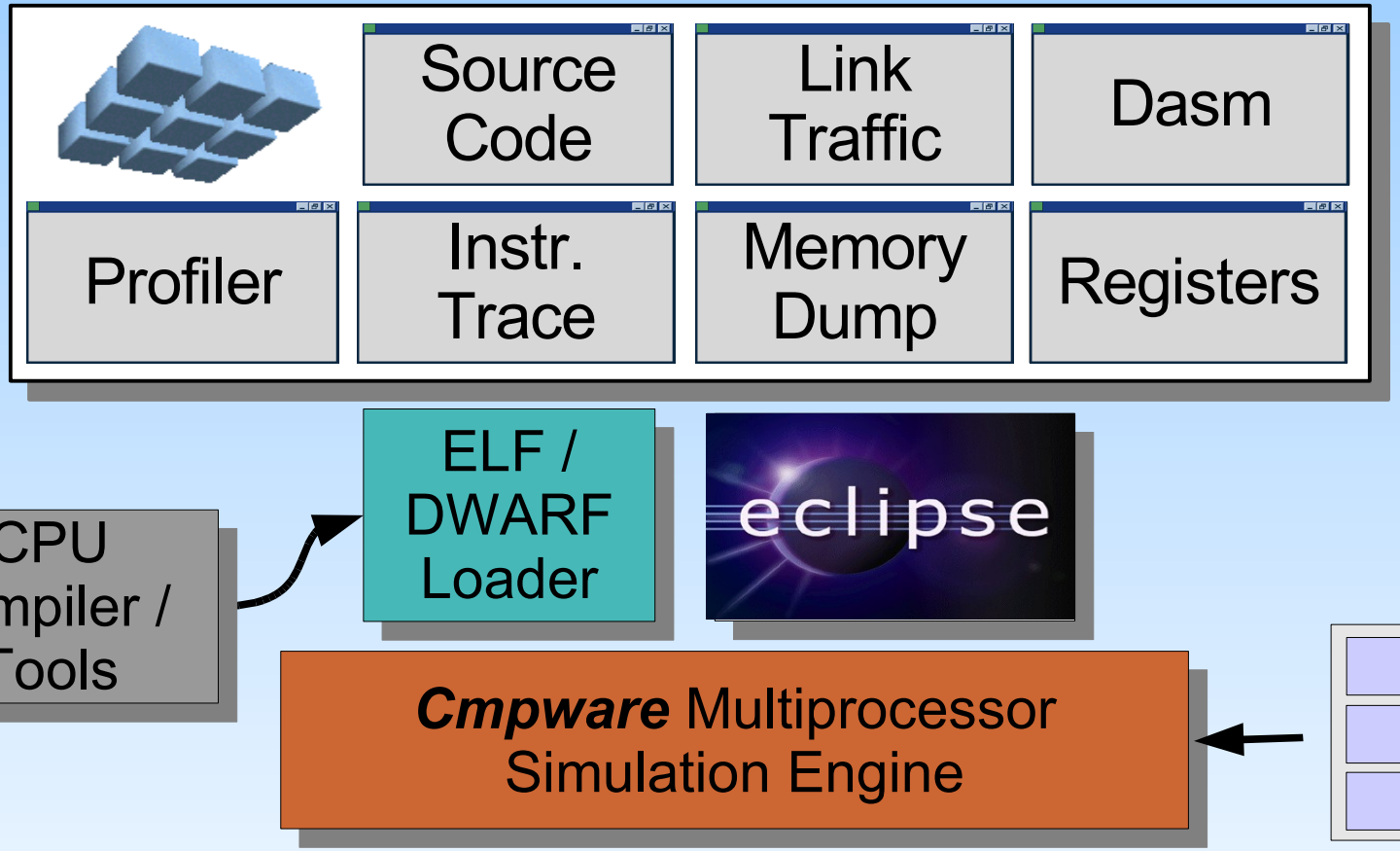




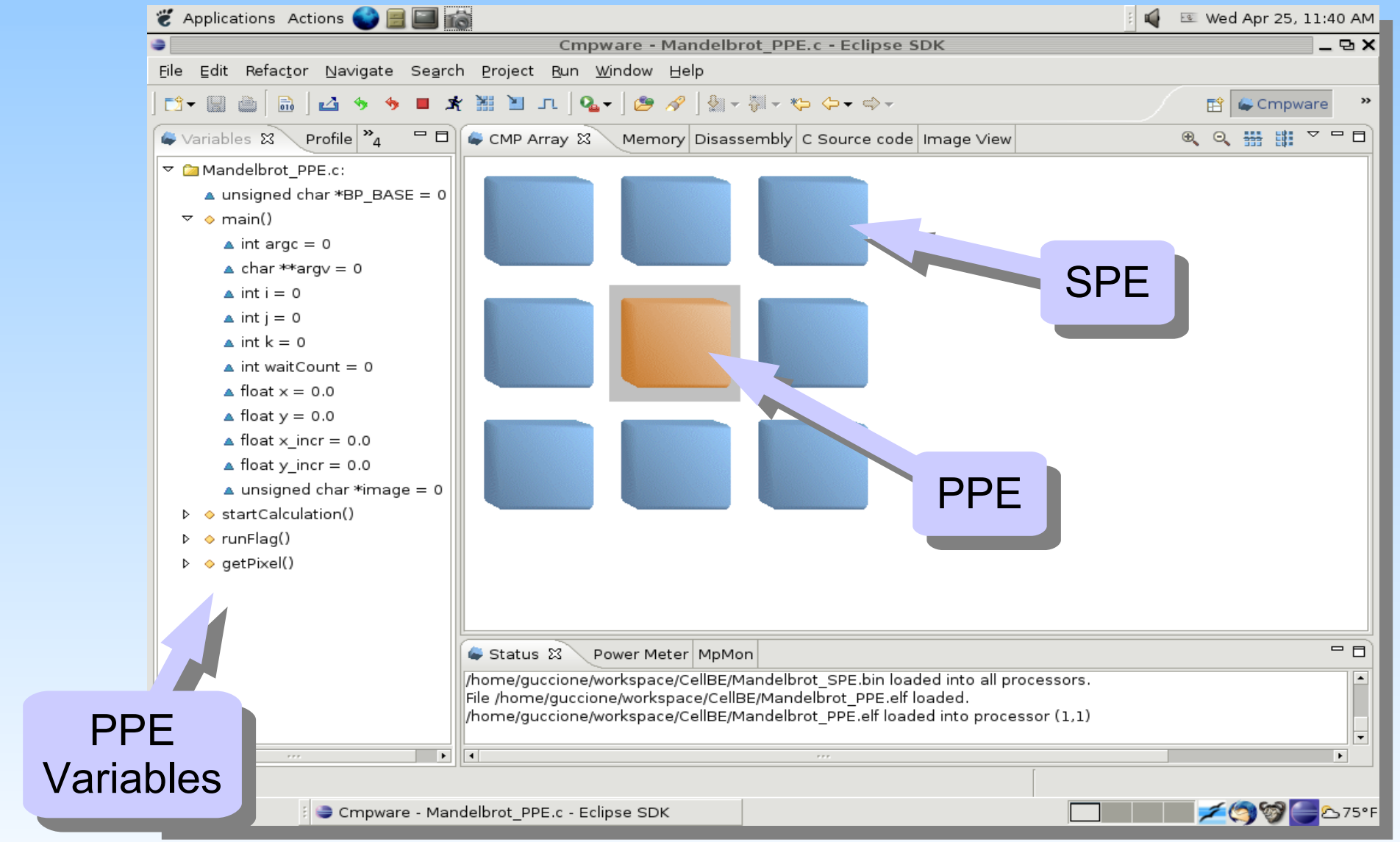
The *Cmpware CMP-DK* IDE

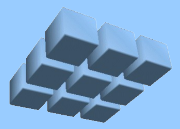
- Multicore simulation model 'plugs in' to the *Cmpware* IDE
- Dynamically customizes the displays for this multicore architecture
- Standard compiled executables run on the simulation model
- A debugger-like interface displays system information, including performance data

Cmpware CMP-DK IDE



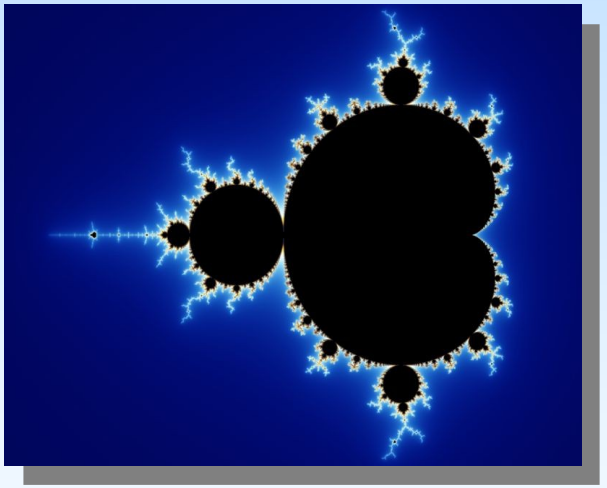
The Cmpware Cell BE Environment

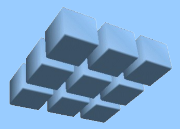




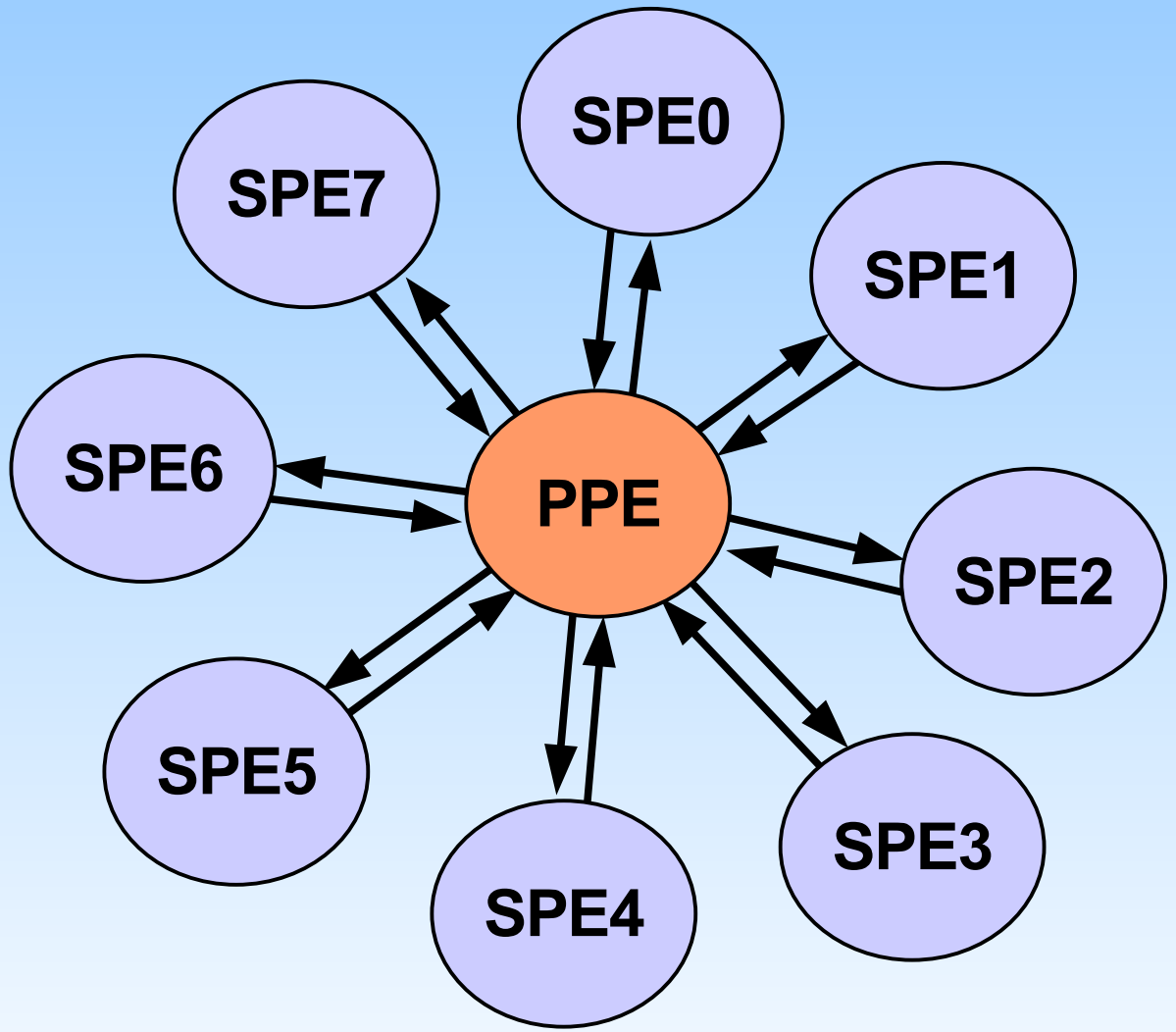
The Mandelbrot Application

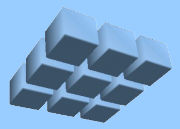
- Non-linear system simulation / analysis
- Floating point intensive, highly parallel
- Demonstrates Cell BE architecture
- Well understood, available benchmarks
- Approach:
  - PPE controls SPEs
  - 'Bare Machine' (no OS)
  - All work done on SPEs





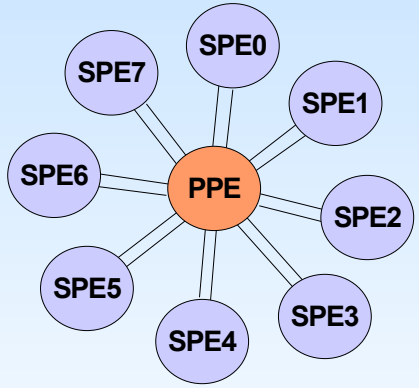
The Computation Model

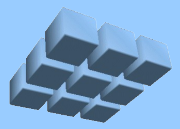




The Computation Model

- General Approach:
  1) Get available SPE
  2) Start new work on available SPE
  3) Repeat until done
- Very simple code
- Can be used for other applications
- Can use subset of SPUs
- Efficient and predictable

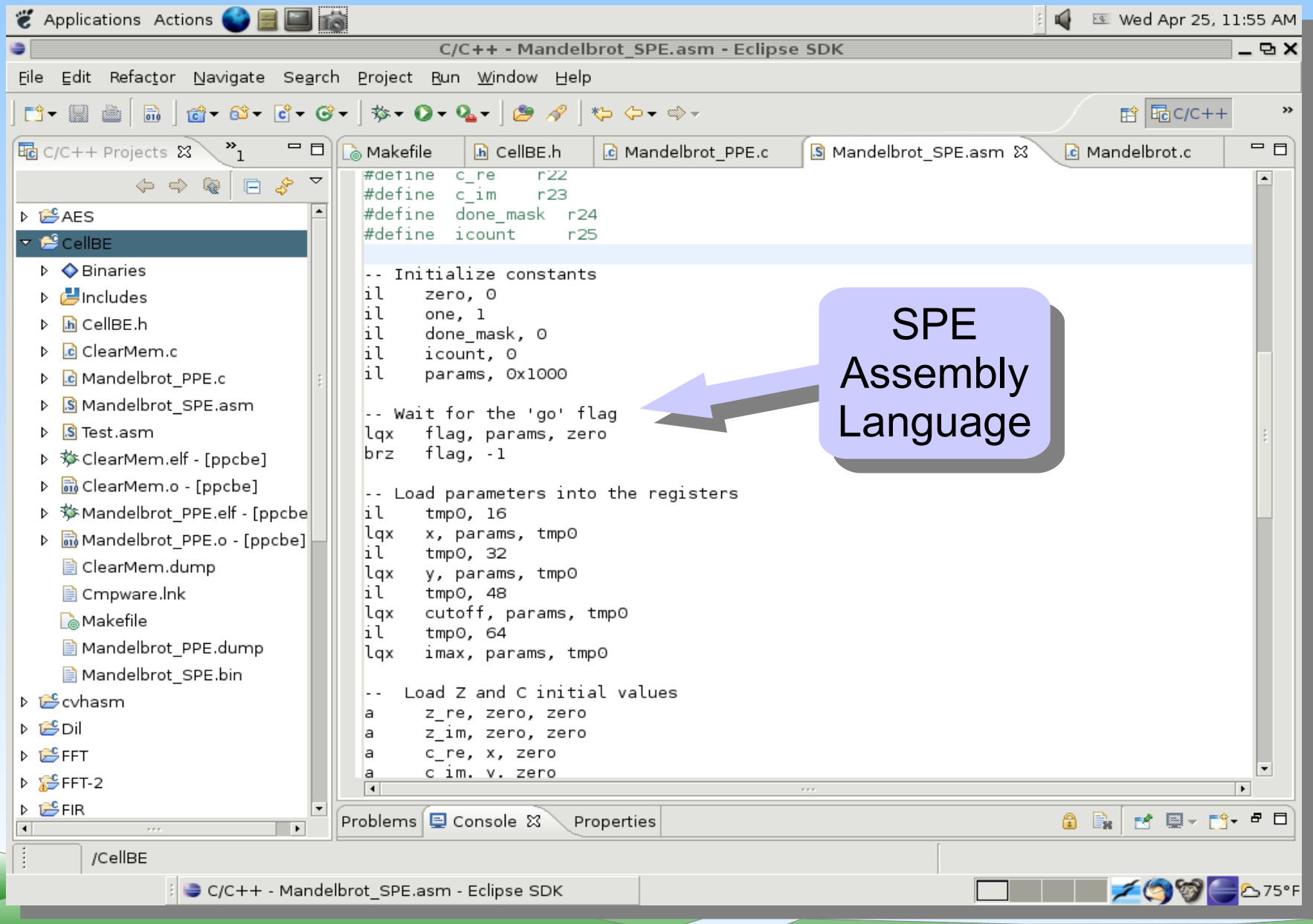




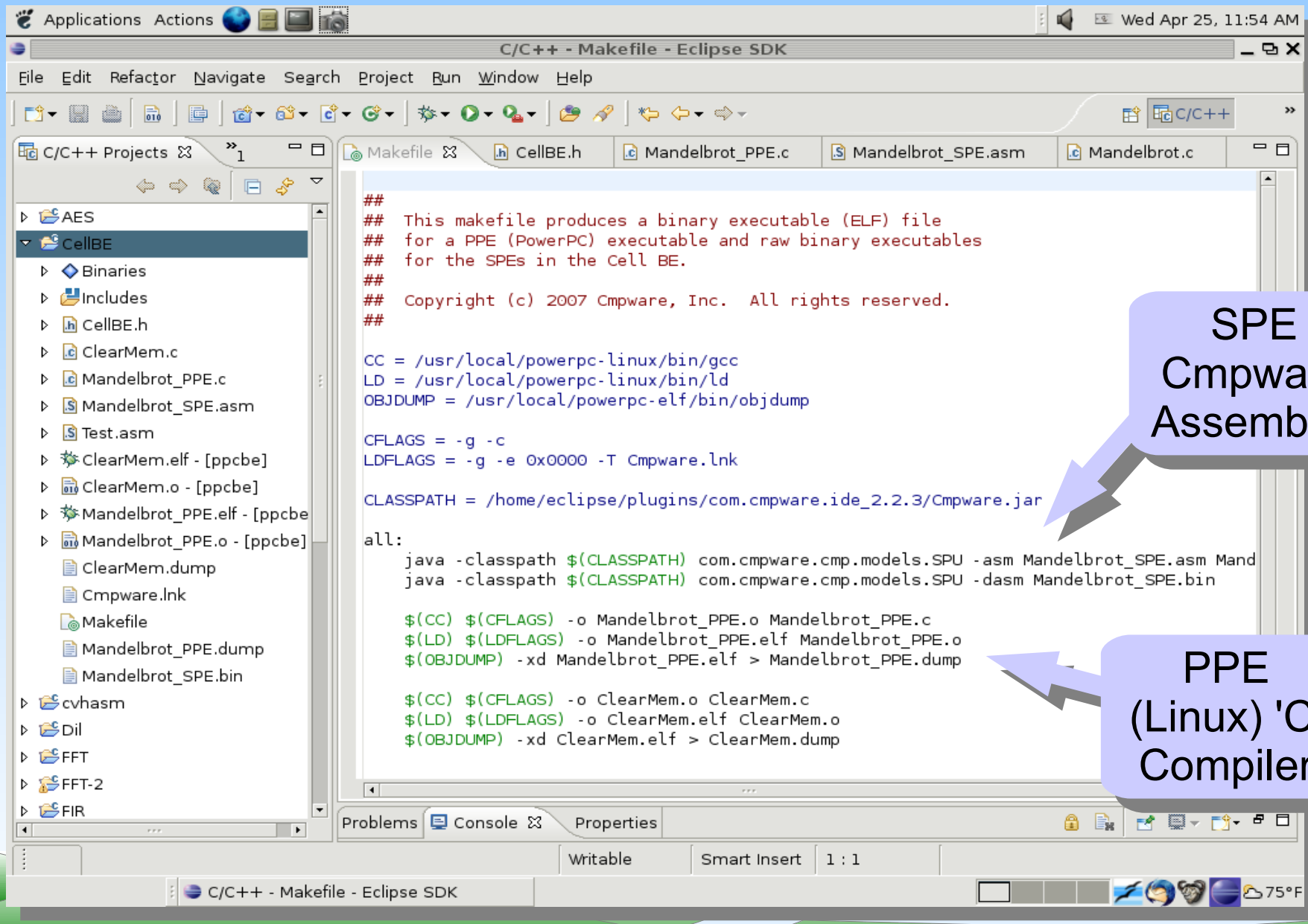
The Mandelbrot Implementation

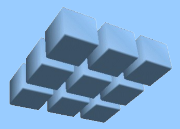
- **PPE**: uses *PowerPC Linux Gnu 'C'* compiler
- **SPE**: uses *'AutoModel'* SPE assembler
- All communication through shared memory
- ***Mandelbrot_PPE.c*** for *PPE*
  - Controls *SPEs*
  - 54 lines of 'C' code (';')
- ***Mandelbrot_SPE.asm*** SPE code
  - 42 lines of *SPE* assembly language

SPE Assembly Code Development



The Makefile





Building SPP and SPE Code

Applications Actions Wed Apr 25, 12:07 PM

C/C++ - Makefile - Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

C/C++ Projects

- AES
- CellBE
  - Binaries
  - Includes
  - CellBE.h
  - ClearMem.c
  - Mandelbrot_PPE.c
  - Mandelbrot_SPE.asm
  - Test.asm
  - ClearMem.elf - [ppcbe]
  - ClearMem.o - [ppcbe]
  - Mandelbrot_PPE.elf - [ppcbe
  - Mandelbrot_PPE.o - [ppcbe]
  - ClearMem.dump
  - Cmpware.lnk
  - Makefile
  - Mandelbrot_PPE.dump
  - Mandelbrot_SPE.bin
- cvhasm
- Dil
- FFT
- FFT-2
- FIR

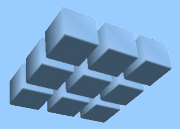
Makefile | CellBE.h | Mandelbrot_PPE.c | Mandelbrot_SPE.asm | Mandelbrot.c

Problems | Console | Properties

C-Build [CellBE]

```
58800914   fa  r20, r18, r0
58858A14   fa  r20, r20, r22
5885CA95   fa  r21, r21, r23
58C50A11   fm  r17, r20, r20
58C54A92   fm  r18, r21, r21
58844911   fa  r17, r18, r17
5842C892   fcgt  r18, r17, r11
08248C18   or  r24, r24, r18
18204C11   and  r17, r24, r1
18044C99   a  r25, r25, r17
0803008C   sf  r12, r1, r12
D4FFF80C   brnz  r12, -16
4080280F   il  r15, 80
4883C719   stqx  r25, r14, r15
48800700   stqx  r0, r14, r0
30000000   bra  0
40200000   nop
40200000   nop

   (42 instructions disassembled).
/usr/local/powerpc-linux/bin/gcc -g -c -o Mandelbrot_PPE.o Mandelbrot_PPE.c
/usr/local/powerpc-linux/bin/ld -g -e 0x0000 -T Cmpware.lnk -o Mandelbrot_PPE.elf
Mandelbrot_PPE.o
/usr/local/powerpc-elf/bin/objdump -xd Mandelbrot_PPE.elf > Mandelbrot_PPE.dump
/usr/local/powerpc-linux/bin/gcc -g -c -o ClearMem.o ClearMem.c
/usr/local/powerpc-linux/bin/ld -g -e 0x0000 -T Cmpware.lnk -o ClearMem.elf ClearMem.o
/usr/local/powerpc-elf/bin/objdump -xd ClearMem.elf > ClearMem.dump
```

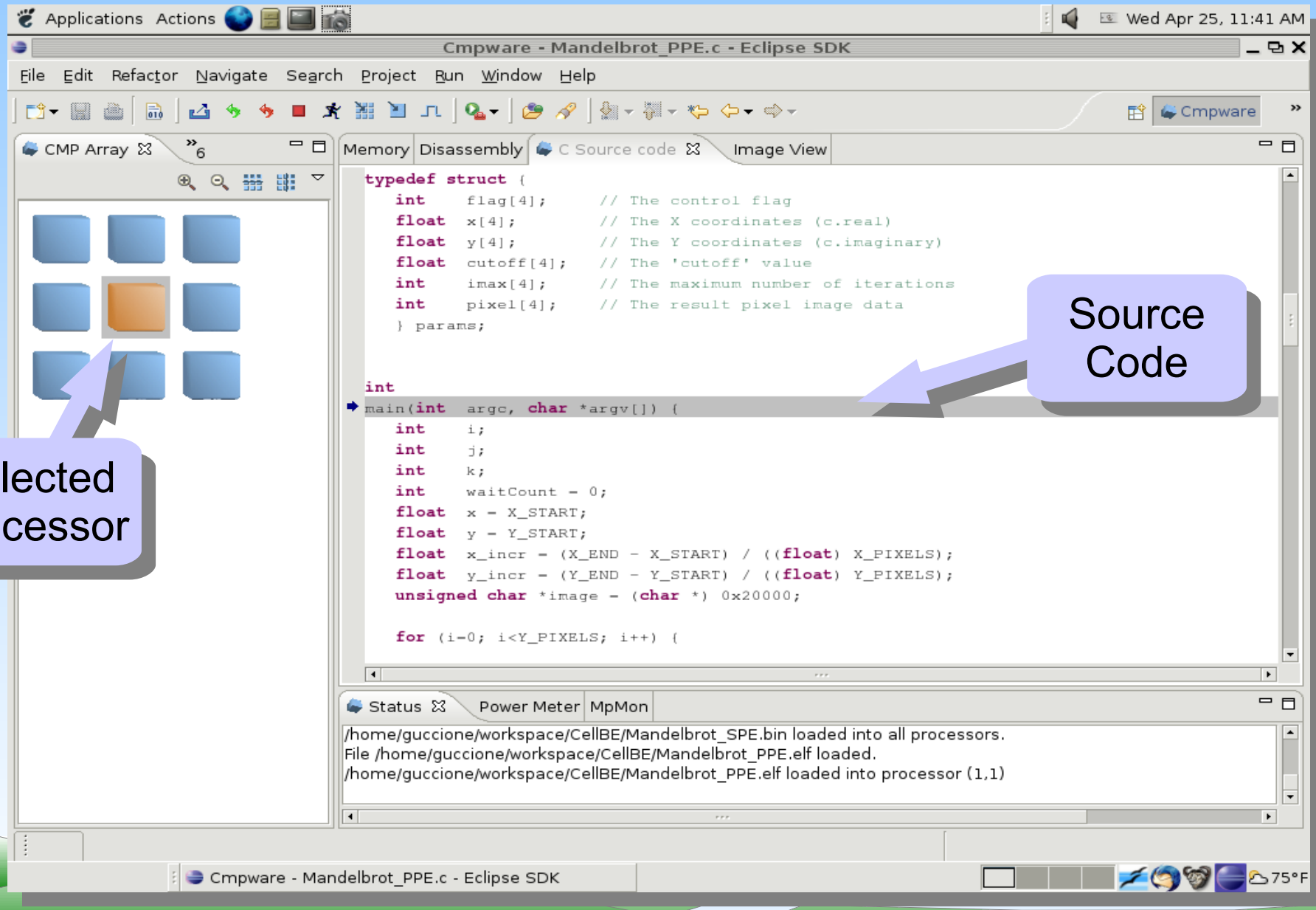
C/C++ - Makefile - Eclipse SDK 75°F



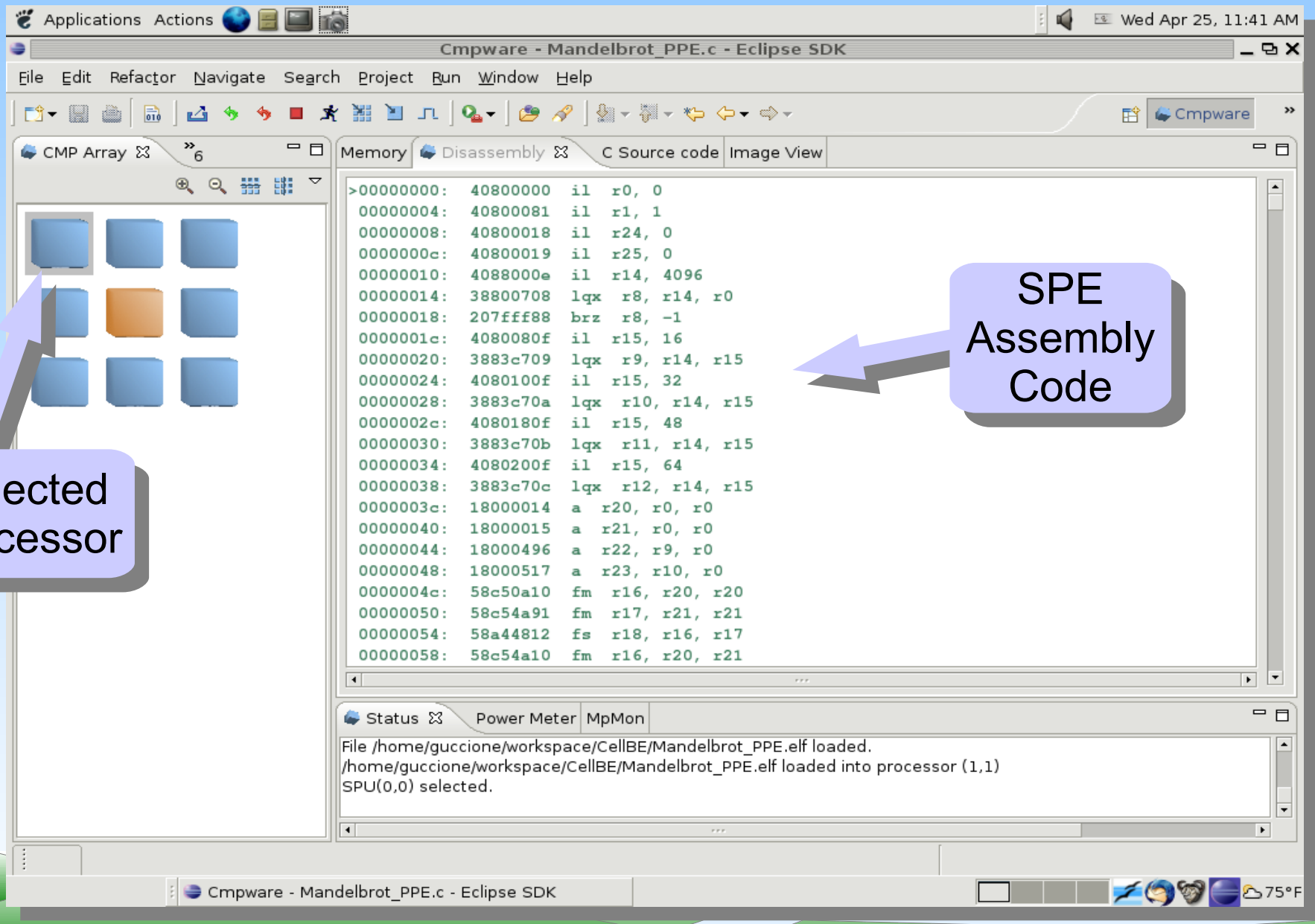
The Cmpware Assemblers

- Cmpware models contain simple assemblers
  - Information extracted from simulation models
  - Supports all processor instructions plus other features (comments, #defines, etc.)
- Very useful in custom architectures
- Demonstrated here for SPE code
  - Only a few instructions required
  - Easy to use
  - No new tools to install

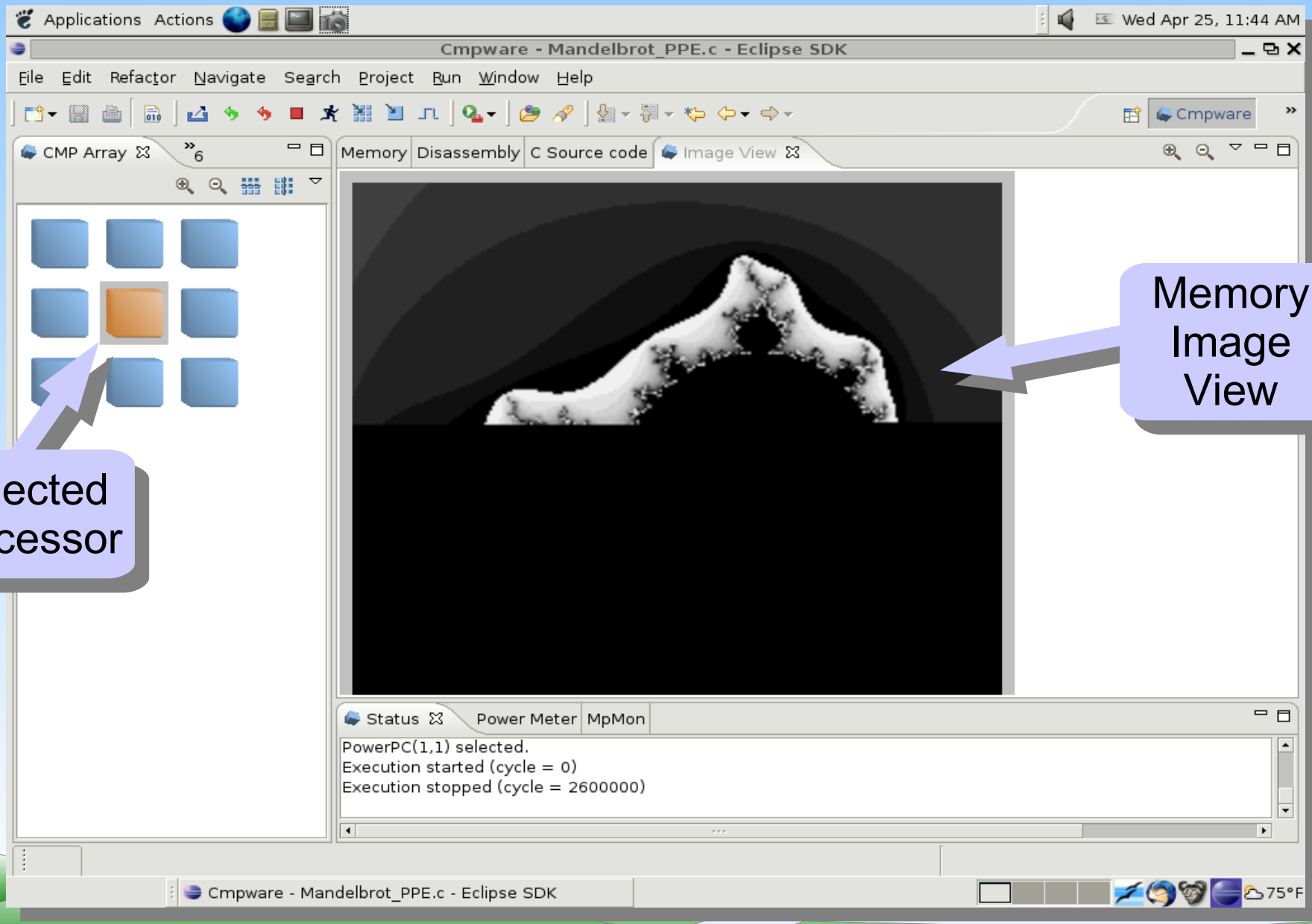
Running the Application

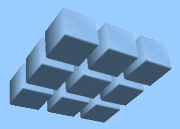


Running the Application



Running the Application



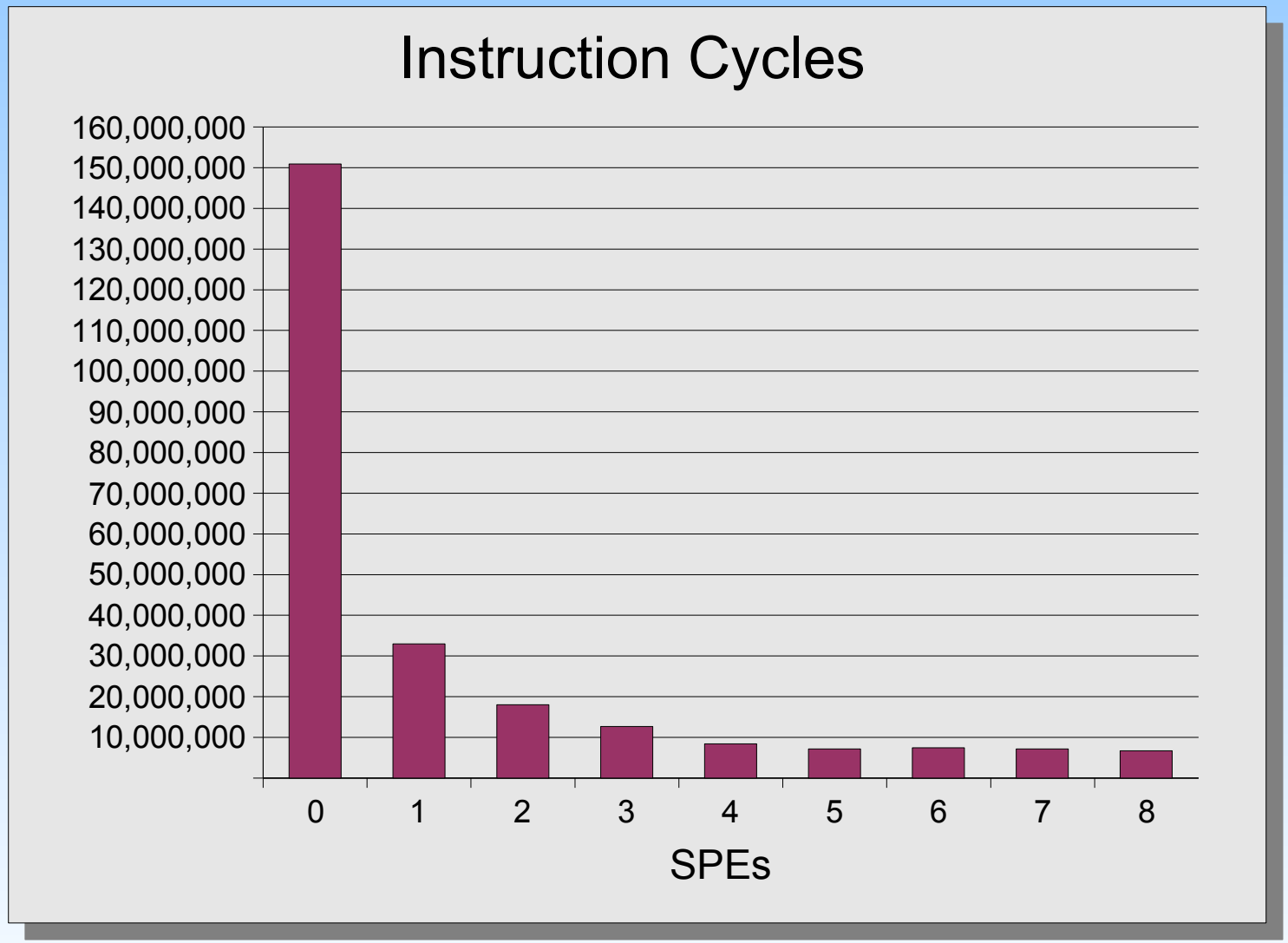


Benchmarking and Performance

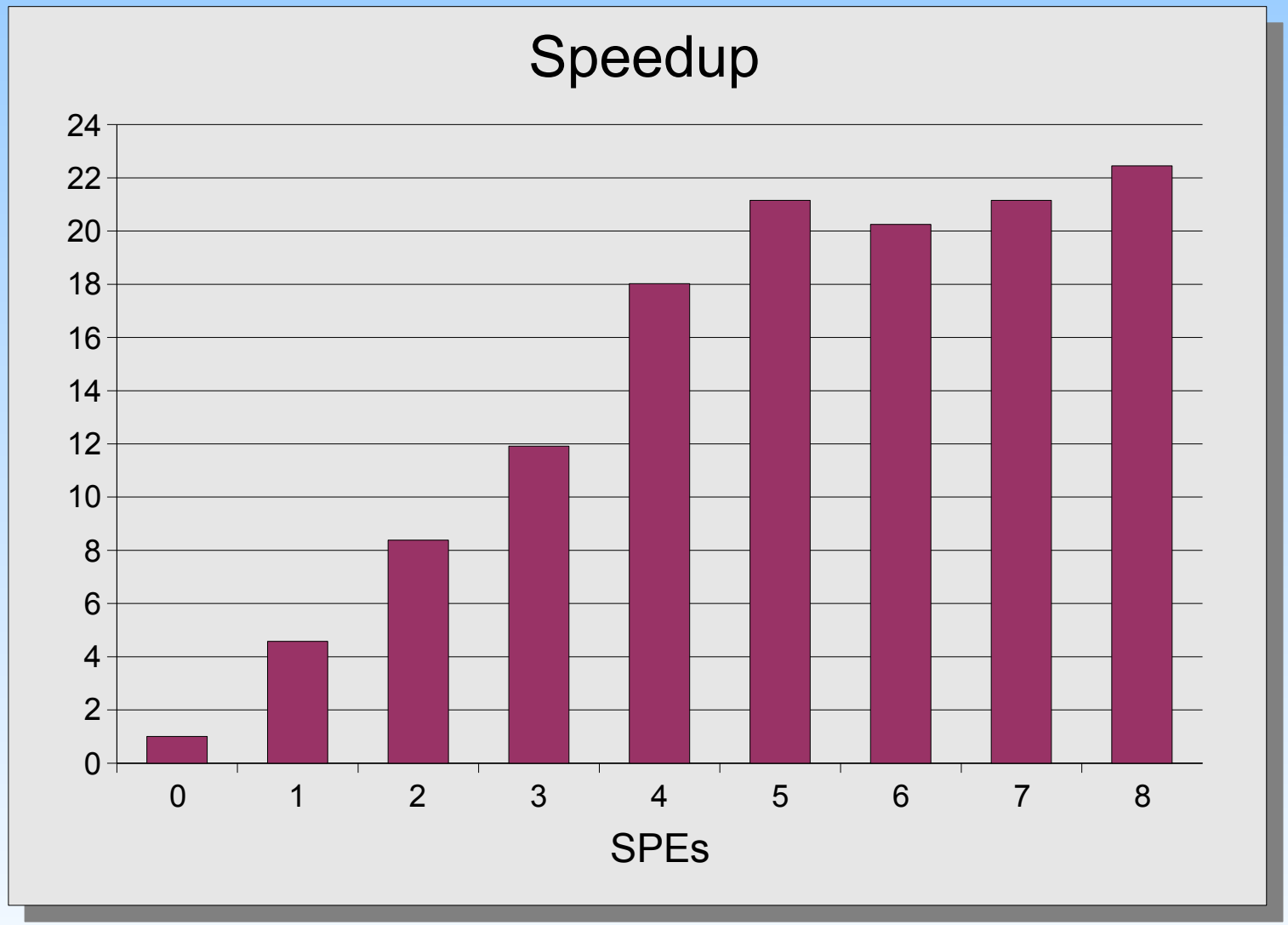
- Cell BE models count instructions
- Not 'cycle accurate'
- Instruction timing can be added to models

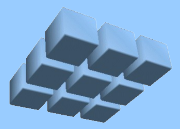
  ... but algorithm partitioning does not need this level of accuracy
- Multiple runs of Mandelbrot algorithm using different numbers of SPUs
- Demonstrates performance boost of SPEs, and overheads involved in parallelizing

Mandelbrot Instruction Cycles



Mandelbrot Speedup





Cell BE Software Development

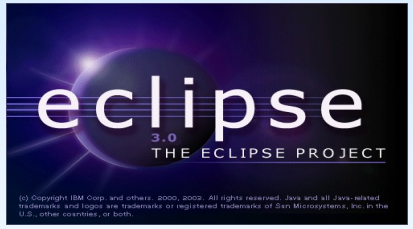
- Edit, compile, execute *and* debug Cell BE software

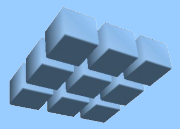
  ... all in the same friendly environment

- Develop Cell BE code faster
- Evaluate Cell BE performance more quickly
- Faster feedback for algorithm partitioning
- Evaluate more alternatives in less time
- Produce more reliable software

Cmpware CMP-DK

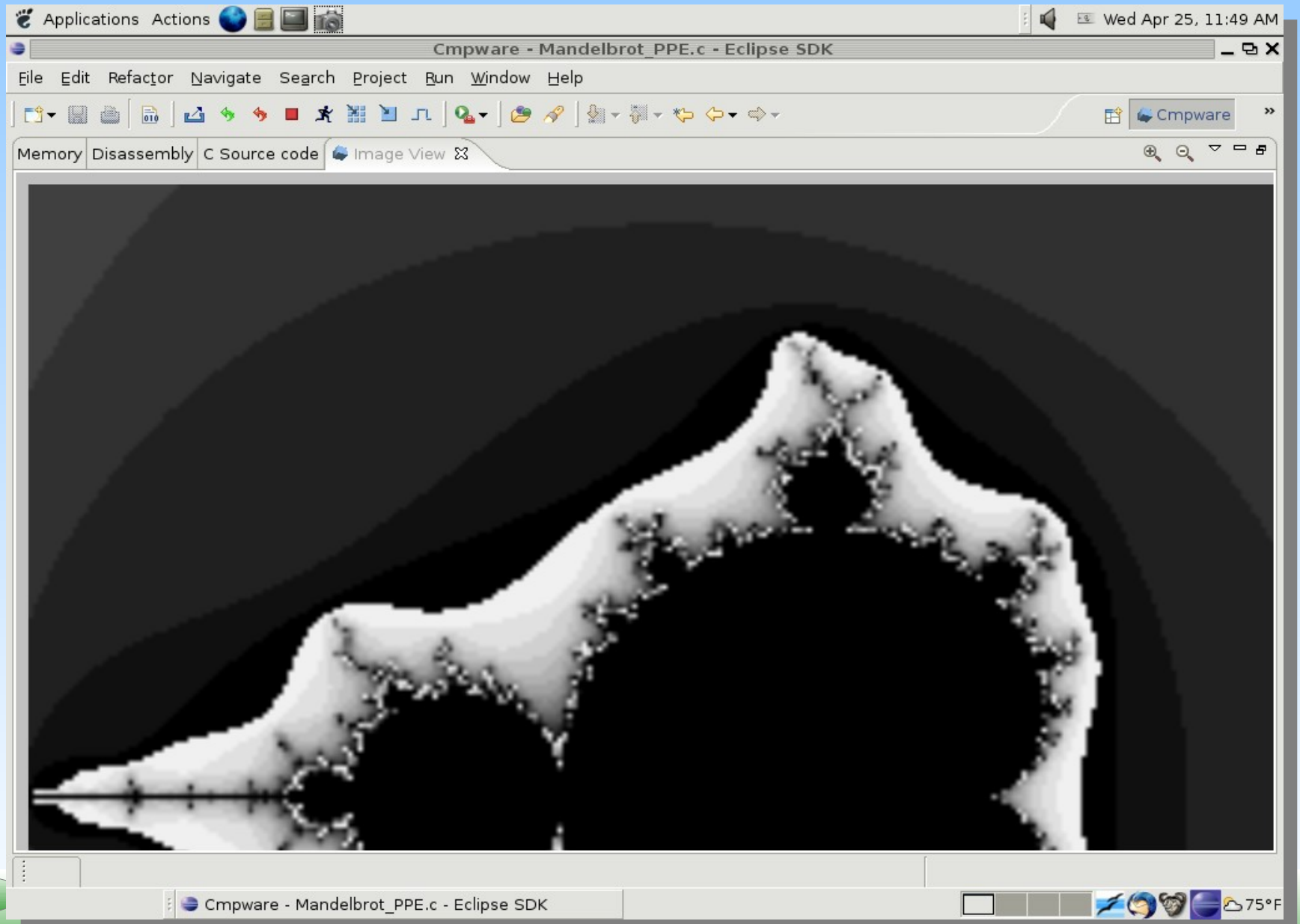
- *Eclipse* / *Java* based
- Runs 'everywhere'
- Completely self-contained
- Compact: 1MB 'plugin'
- Easy to install (seconds)
- Our goal: *to make multicore software development easier*

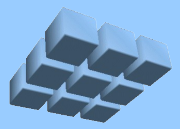




Extra Slides

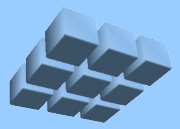
Image View (Maximized)





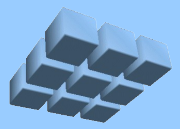
CellBE.h Header file

```
/*
**
**  This defines the shared memory in the Cell BE processor.
**
**  Copyright (c) 2007 Cmpware, Inc.  All rights reserved.
**
*/

#ifndef CELLBE_H_
#define CELLBE_H_


/* A shared memory address */
typedef unsigned char *Address;

/** The number of SPEs */
#define  SPES  8

/* The size of the SPE local memory */
#define  SPE_MEMORY_SIZE   (16 * 1024)

/* The range of memory occupied by an SPE in the PPE memory map */
#define  SPE_MEMORY_RANGE  (16 * 1024)

/* The start of the SPE shared memory */
Address  BP_BASE = (Address) (256 * 1024);

#endif /* CELLBE_H_*/
```



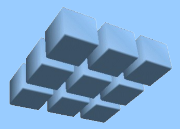
The PPE Inner Loop Code

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

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

        /* Get pixel results */
        for (k=0; k<SPES; 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) */
```



PPE Shared Memory Code

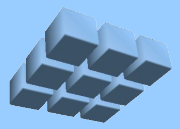
```
void  startCalculation(int  spe, float x, float y, float x_incr) {
   params *p = (params *) (BP_BASE + (spe * SPE_MEMORY_RANGE) + 0x1000);
   p->x[0] = x;
   p->x[1] = x + x_incr;
   p->x[2] = x + (2 * x_incr);

  [...]

   p->flag[3] = 0;
   p->flag[0] = SPU_BUSY;  // Start calculation
   }  /* end startCalculation() */


int runFlag(int spe) {
   params *p = (params *) (BP_BASE +(spe * SPE_MEMORY_RANGE) + 0x1000);
   return (p->flag[0]);
   }  /* end runFlag() */


unsigned char getPixel(int  spe, int  pixelNum) {
   params *p = (params *) (BP_BASE + (spe * SPE_MEMORY_RANGE) + 0x1000);
   return ((unsigned char) ((p->pixel[pixelNum]& 0x0f) << 4));
   }  /* end getPixel() */
```



SPE 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
```



SPE Assembly Code

```
--  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
```