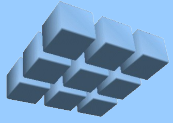


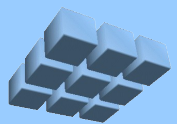
An Image Processing Demonstration Using the Cmpware CMP-DK

Steven A. Guccione
Cmpware, Inc.

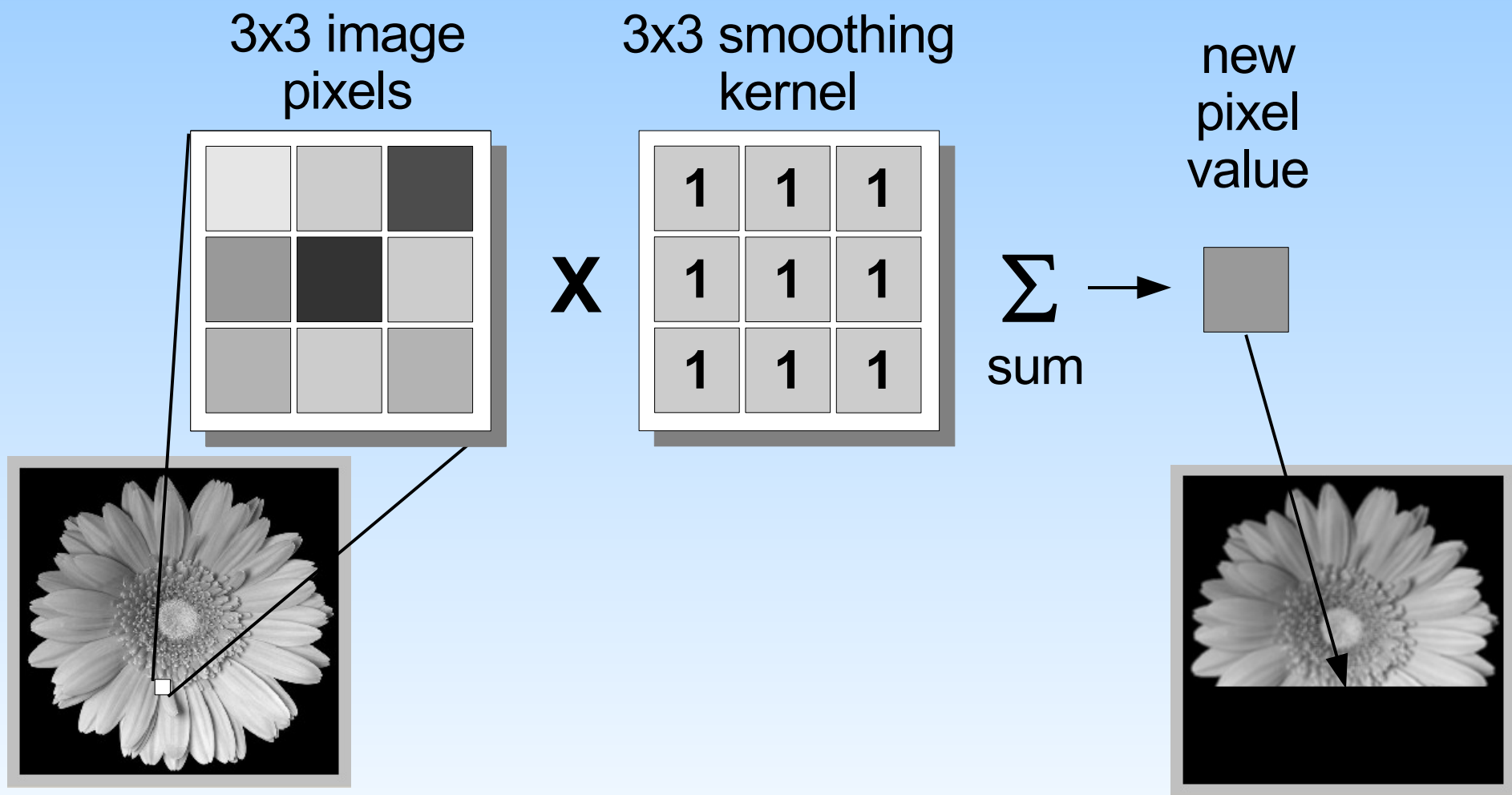


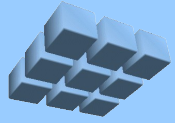
Multicore Image Processing

- Multicore devices increasingly used for image and video processing
- High performance, low power and high levels of programmability make multicore attractive
- This demo uses the *Cmpware CMP-DK* to:
 1. Model a multicore architecture
 2. Write software for this architecture
 3. Execute compiled code on the model
 4. View the results interactively in the IDE



Introduction: Image Morphology





Other Morphology Kernels

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Smoothing

| | | |
|----|----|----|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

Laplacian I

| | | |
|----|----|----|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

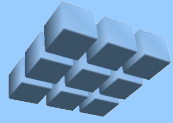
Laplacian II

| | | |
|---|---|----|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

Sobel Horizontal
Edge Detection

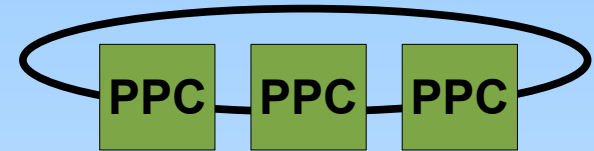
| | | |
|----|----|----|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

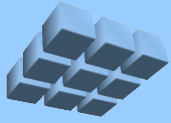
Sobel Vertical
Edge Detection



The Multicore Architecture

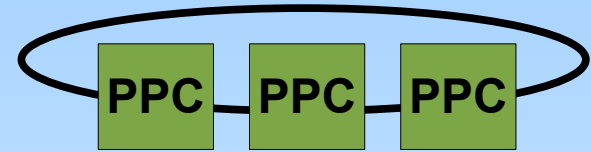
- 3 *PowerPC* processors
- '*Ring*' architecture
- 16k local memory per CPU
 - Image processing code
 - Local data
- 128k shared memory per CPU
 - Shared between adjacent pairs of processors
 - Provides all communication and synchronization



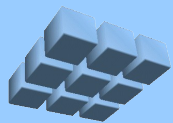


Modeling the Architecture

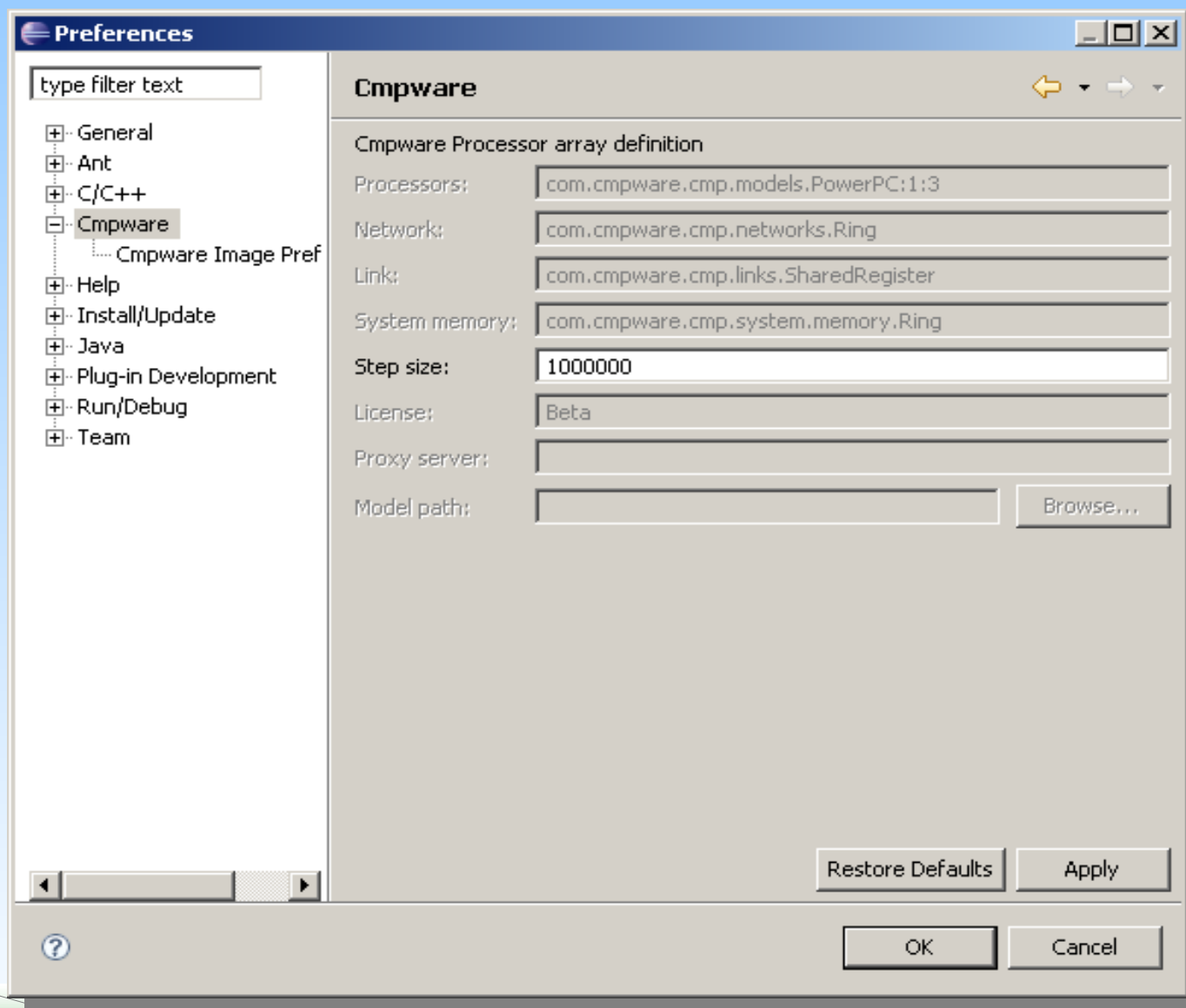
- Uses existing *Cmpware* models:
 - Processor: *PowerPC*
 - Network: *Ring*
 - Link: *SharedMemory* (not used)
 - System Memory: *Ring*
- Pre-configured in '*ppcdemo*' Eclipse plugin installable from:

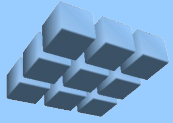


<http://www.cmpware.com/ppcdemo/>

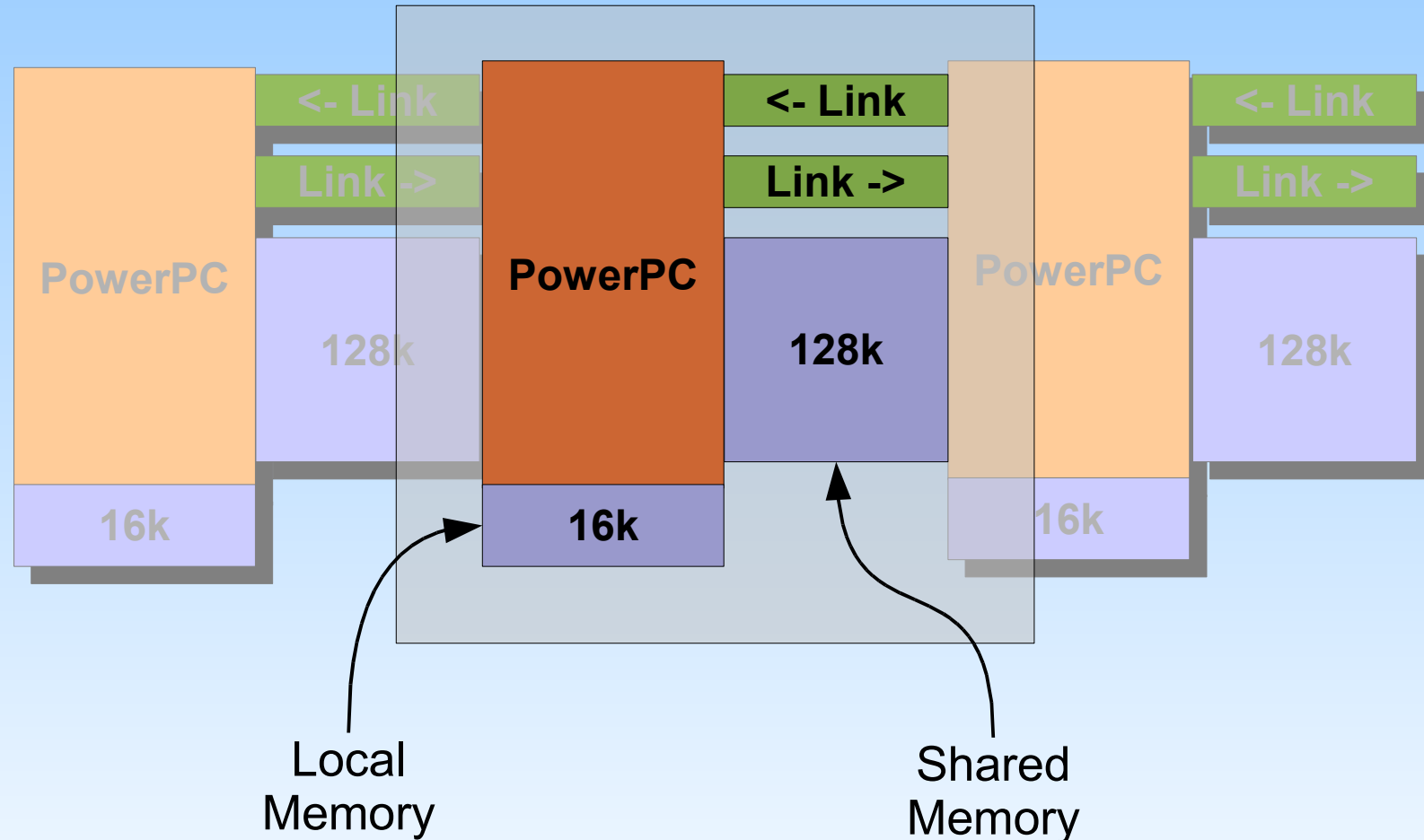


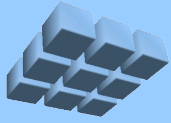
The 'ppcdemo' Preferences



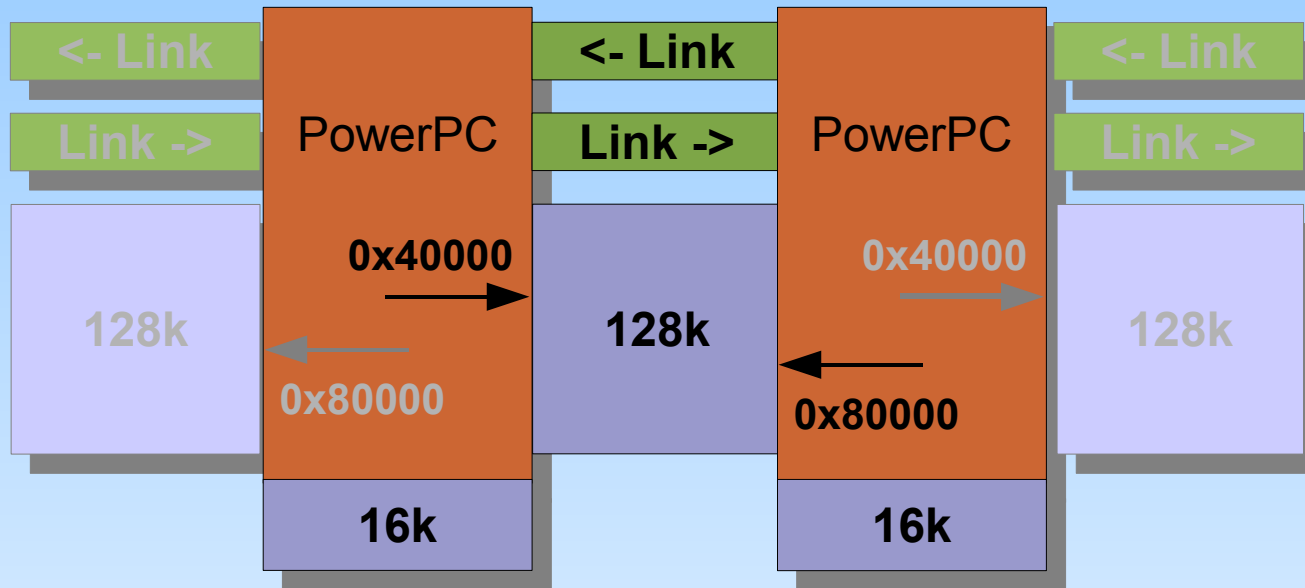


The Processing Node

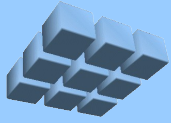




Addressing Shared Memory

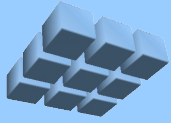


- Each node 'sees' two shared memories
- '*East*' shared memory at address 0x40000
- '*West*' shared memory at address 0x80000

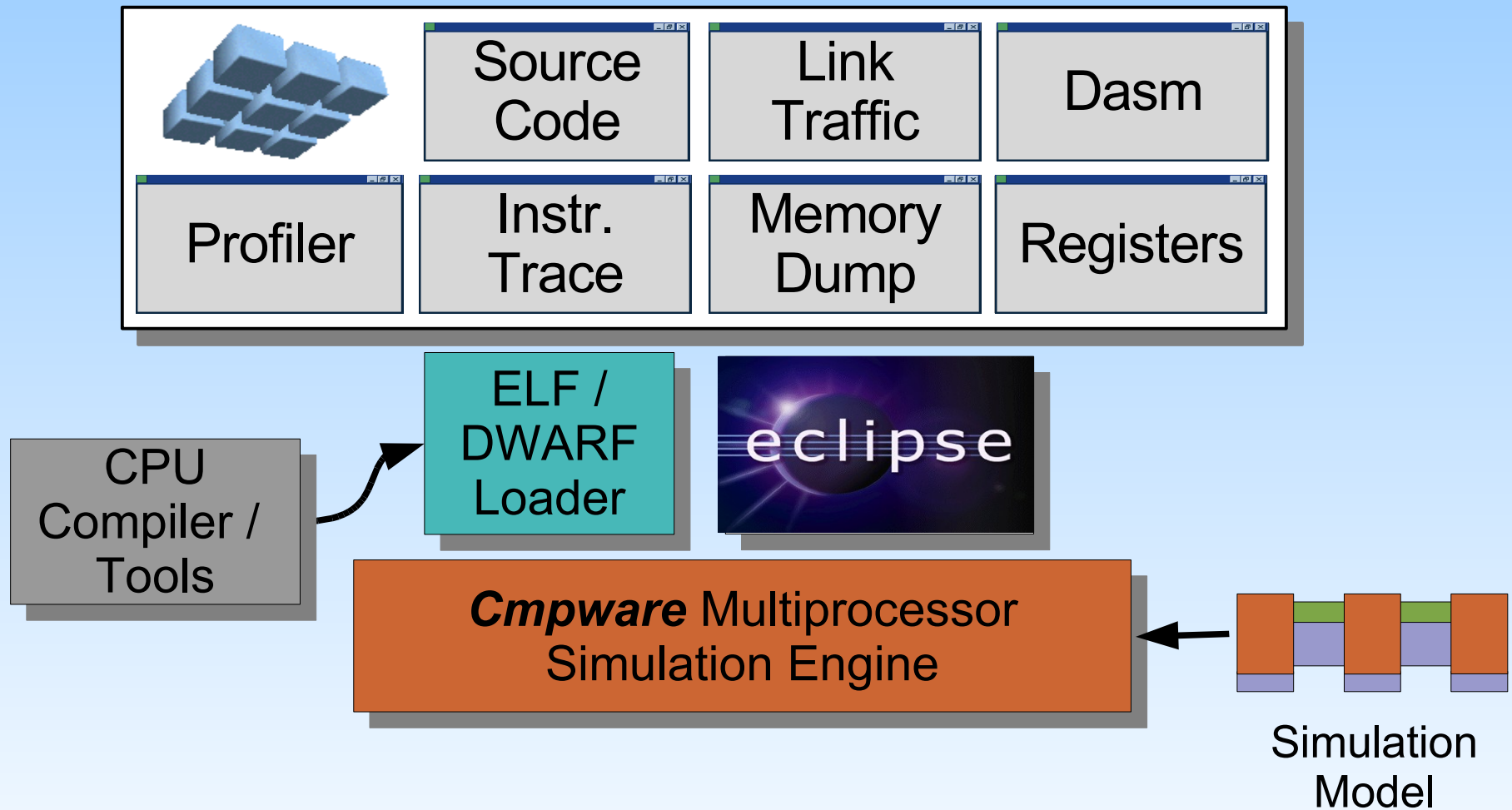


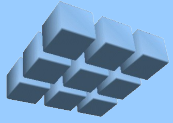
The *Cmpware CMP-DK* IDE

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



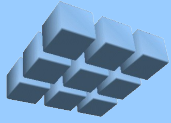
Cmpware CMP-DK IDE





Executing the Application

- Uses existing 'C' compilers (Gnu)
- Communication through shared memory
 - Requires use of same memory map / addresses as hardware simulation model
 - Memory mapped *channels* also available
- ***Image.c*** compiled for different filters
- ***Image0.c*** initiates execution
- Image data pre-loaded into processor (0,0) shared memory from ***Flower256.elf***



Building the 'C' Code

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development. The left sidebar shows the 'C/C++ Projects' view with a tree structure for the 'Image' project, including binaries (Edge.elf, Image0.elf, Smooth.elf), includes (CmpwareRing.h, Image.c, Image0.c), and object files (Edge.o, Image0.o, Smooth.o). The main editor window displays the 'Image.c' source file with the following code:

```
/*
** This program is used for to process an image
** in the Cmpware environment. A command-line define flag
** such as "-DKERNEL=laplacian2" is used to set the kernel type.
**
** Copyright (c) 2007 Cmpware, Inc. All rights reserved.
**
*/

#include "CmpwareRing.h"

char filter(int x, int y, int kernel[3][3]);
int getPixel(int x, int y);

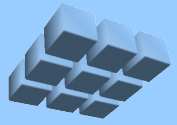
/* The smoothing kernel */
static int smooth[3][3] = {{1,1,1}, {1,1,1}, {1,1,1}};

/* The Laplacian kernel */
static int laplacian[3][3] = {{-1,-1,-1}, {-1,8,-1}, {-1,-1,-1}};

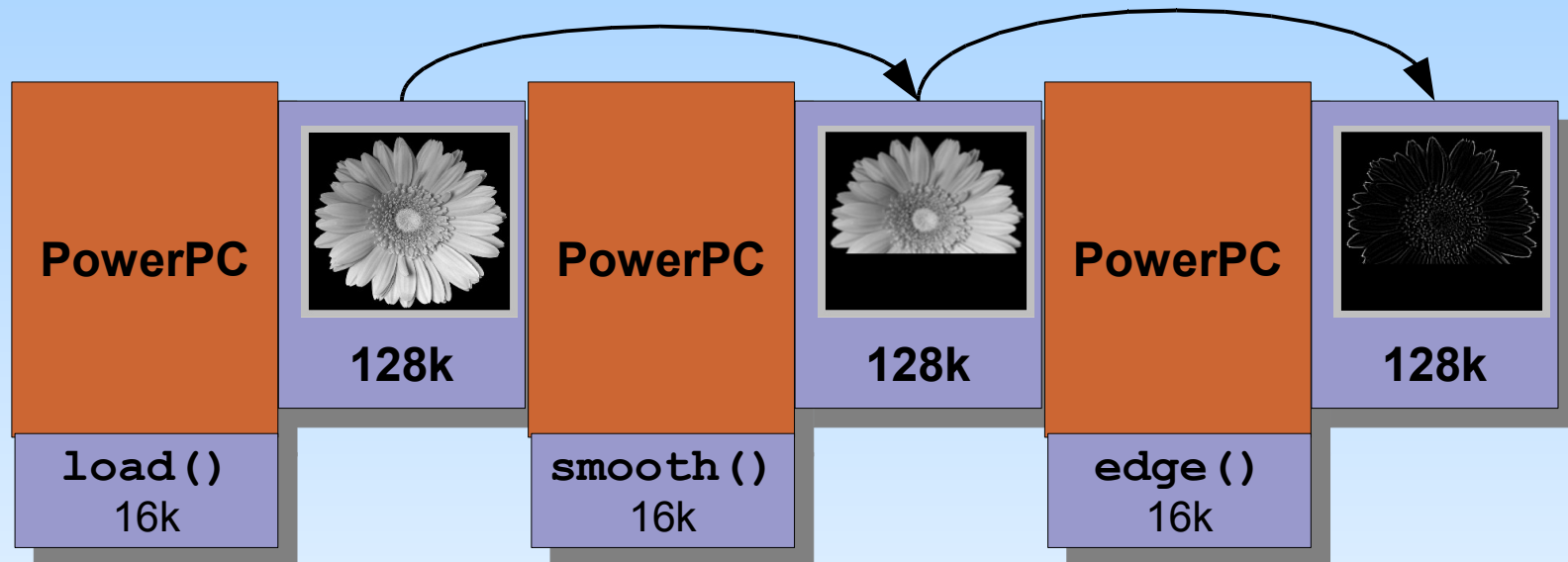
/* Another Laplacian kernel */
static int laplacian2[3][3] = {{0,1,0}, {1,4,1}, {0,1,0}};
```

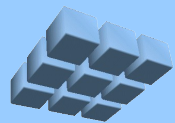
The bottom panel shows the 'Problems' view with the following build output:

```
C-Build [Image]
/usr/local/powerpc-linux/bin/ld -g -e 0x0000 -T Cmpware.lnk -o Smooth.elf Smooth.o
/usr/local/powerpc-elf/bin/objdump -xd Smooth.elf > Smooth.dump
/usr/local/powerpc-linux/bin/gcc -g -c -o Edge.o -DKERNEL=laplacian2 Image.c
/usr/local/powerpc-linux/bin/ld -g -e 0x0000 -T Cmpware.lnk -o Edge.elf Edge.o
/usr/local/powerpc-elf/bin/objdump -xd Edge.elf > Edge.dump
```

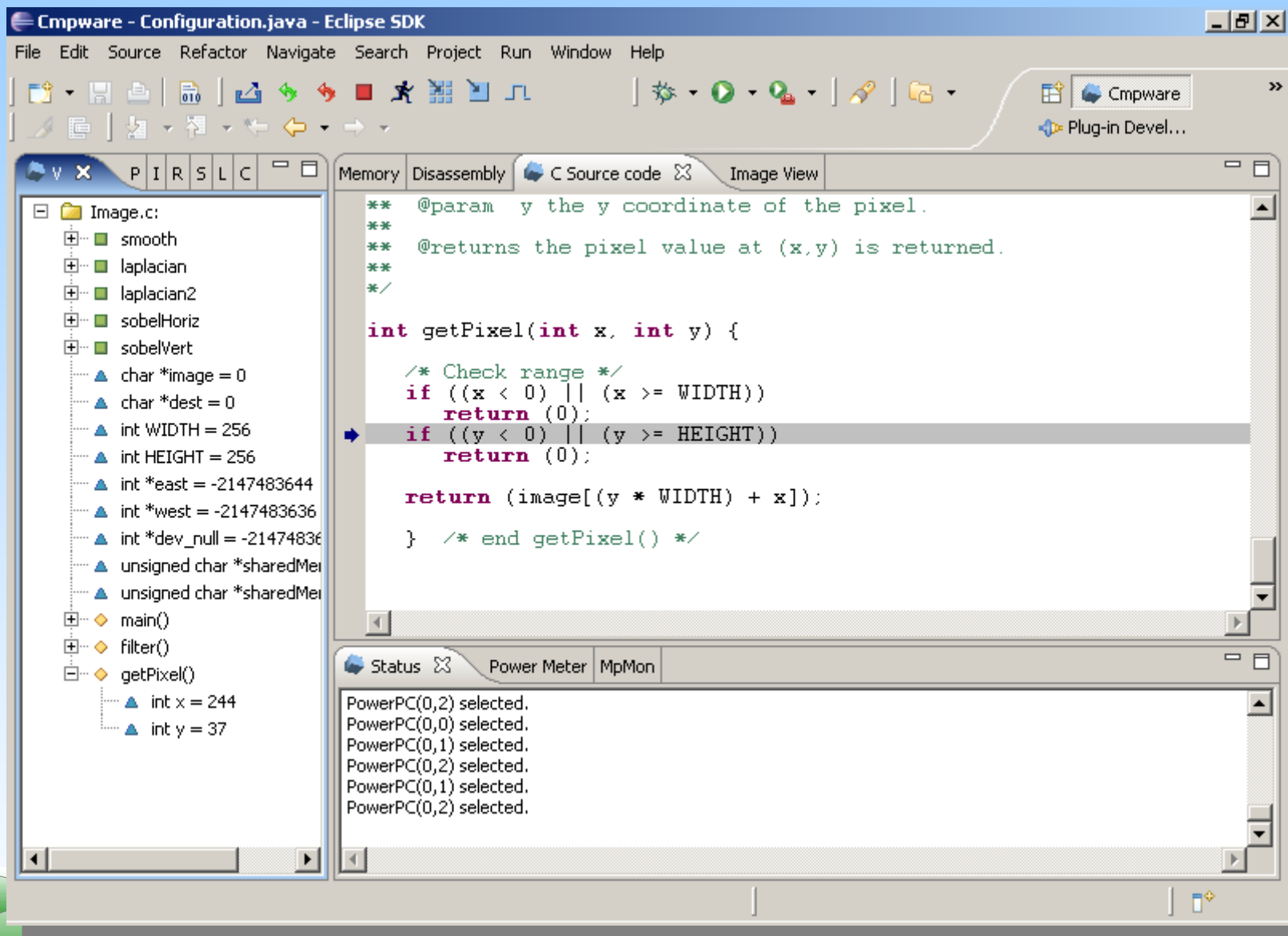


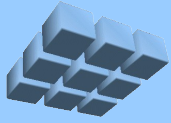
The Image Filtering Application





Running the 'C' Code in the *Cmpware CMP-DK*



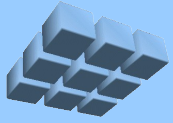


The Inner Loops

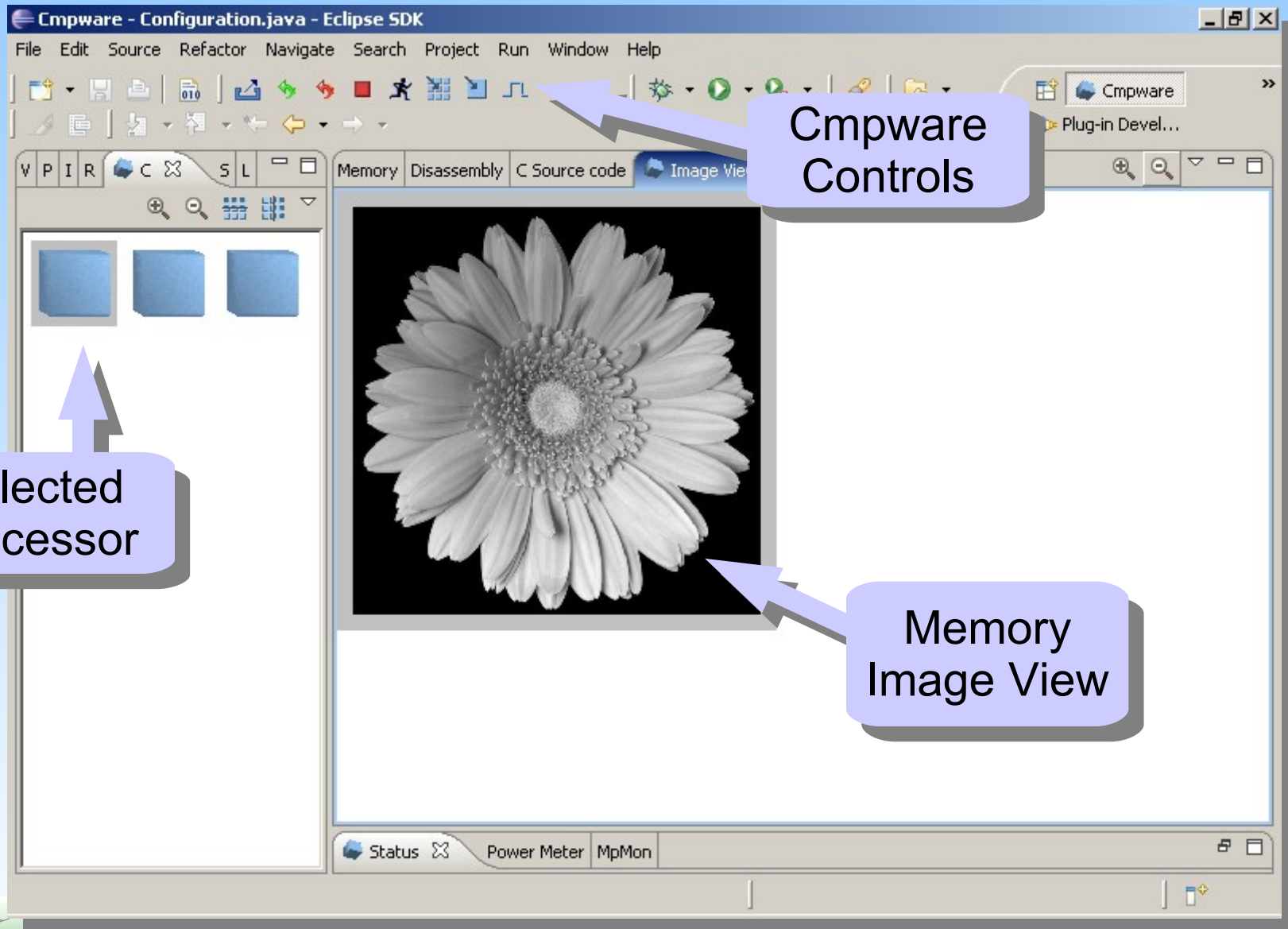
```
/* Wait until image data available */
while (*(image+(HEIGHT*WIDTH)) == 0)
    ;    // wait

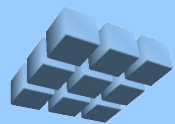
/* Process the image */
for (y=0; y<HEIGHT; y++)
    for (x=0; x<WIDTH; x++)
        dest[(y*WIDTH)+x] = filter(x, y, KERNEL);

/* Synchronize */
/* (tell the next processor that the image is ready) */
*(dest + (HEIGHT*WIDTH)) = 1;
```

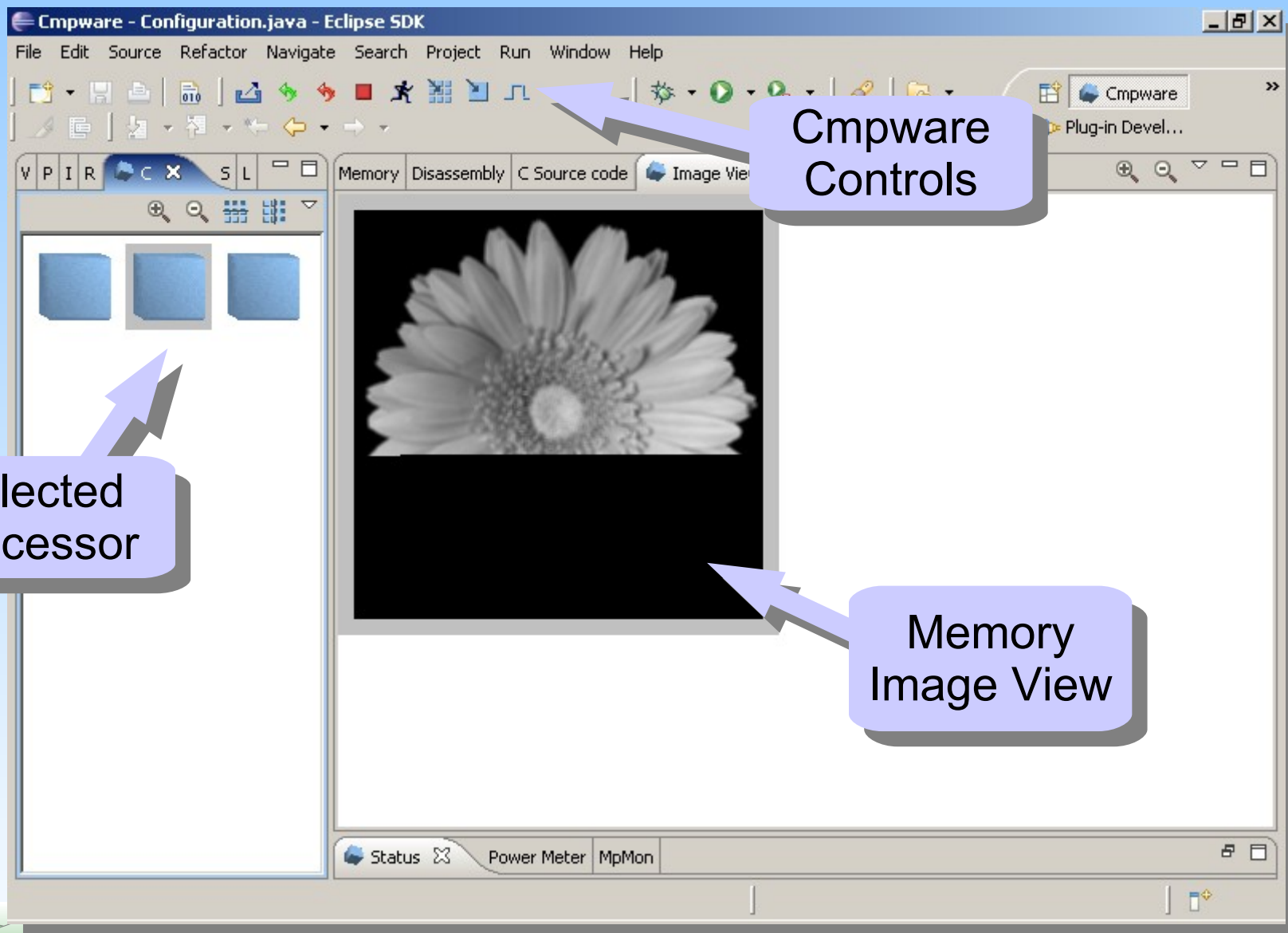



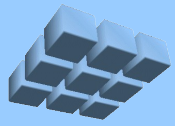
The P(0,0) Original Image



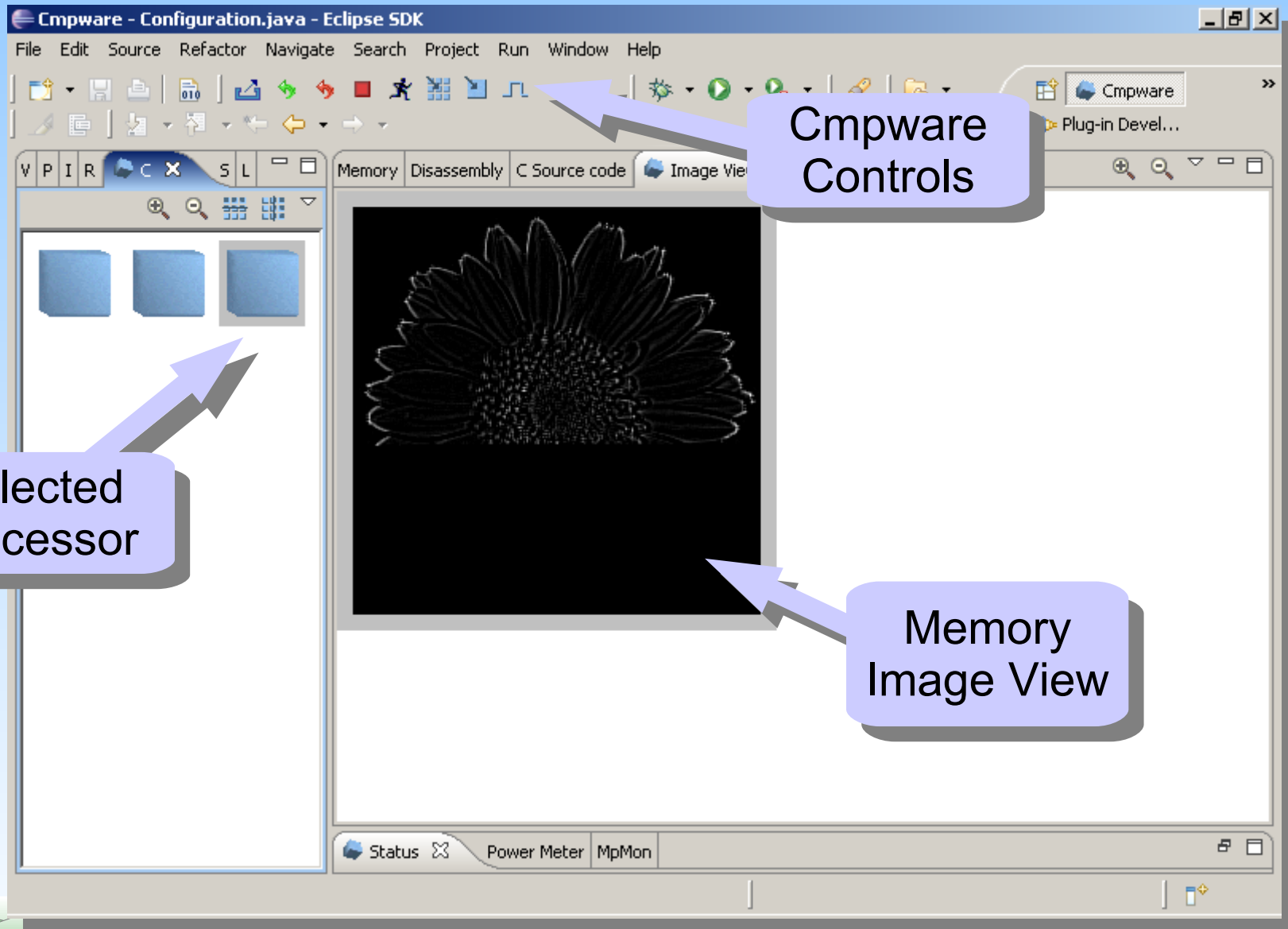


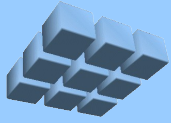
The P(1,0) Smoothed Image





The P(2,0) Edge-detected Image



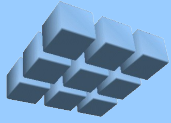


Performance

- Processor (0,0) only loads original image
- Two processors processing images
 - Smoothing with 3 x 3 kernel
 - Edge enhancement with 3 x 3 Laplacian filter
- **100M** cycles total execution

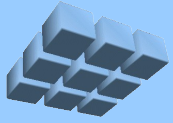
Problem: edge detection waits for entire image smoothing before beginning

==> 50% processor utilization (even / odd pattern)



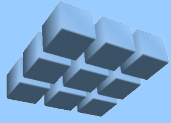
Improving Performance

- Performance limited by synchronization
- No need to wait for entire image
- **Plan:** synchronize at every line of pixels
 - Somewhat more complex code
 - Executes in **50M** cycles
 - **2x** performance of original code
 - Approaches **100%** processor efficiency
 - Performance extends to higher numbers of processors



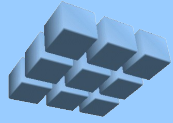
The Improved Inner Loops

```
/* Process the image */  
for (y=0; y<HEIGHT; y++) {  
  
    while (*imageLineReady < (y+3))  
        ; // wait for data to be ready  
  
    /* Process line of pixels */  
    for (x=0; x<WIDTH; x++)  
        dest[(y*WIDTH)+x] = filter(x, y, KERNEL);  
  
    /* Tell the next processor we finished line <y> */  
    *currentImageLine = y;  
  
} /* end for(y) */
```



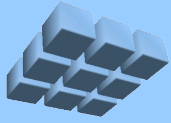
Parallelism

- Lots of parallelism available in this algorithm
- Every stage depends on 3 available lines
- Each pixel can be computed in parallel
- Potential for hundreds of processing cores
- Real-time requirements suggest far fewer
 - 1k x 1k video at 30 fps = 30M pixels / sec
 - At 100 ops per pixel, 3B ops / sec
 - 10 cores at 300 Mhz (approx.)



A Note on Synchronization

- Proper synchronization very important
- Shared Memory generally requires atomic *'test-and-set'* operation
- This Image Processing algorithm:
 - Only sends data in one direction
 - One reader, one writer
 - Can use a simpler synchronization scheme
- *Cmpware* **'channels'** suitable for more general-purpose synchronization

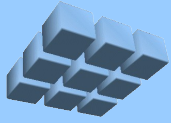


Cmpware CMP-DK

- Model complex multicore processors
- Edit, compile, execute *and* debug multicore software

... all in the same friendly environment

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



Installing the Demo

- Available as an *Eclipse* plugin at the Eclipse update site:

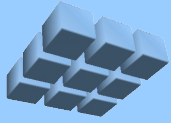
<http://www.cmpware.com/ppcdemo/>

- Other files available at:

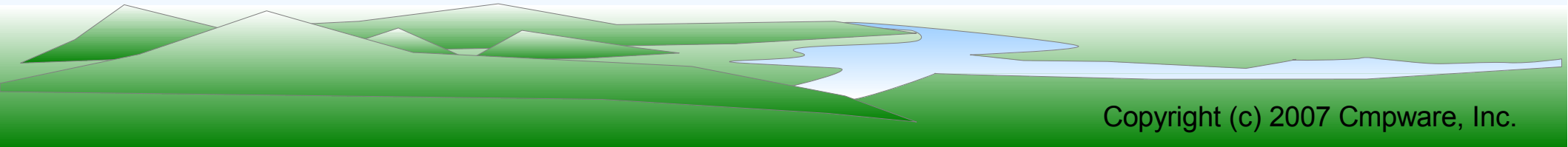
<http://www.cmpware.com/ppcdemo/Ppcfiles.zip>

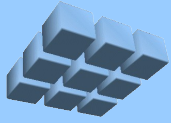
- For more information on how to install an *Eclipse* plugin from an update site, see:

http://www.cmpware.com/demo/DemoInstall_2.2.1.pdf

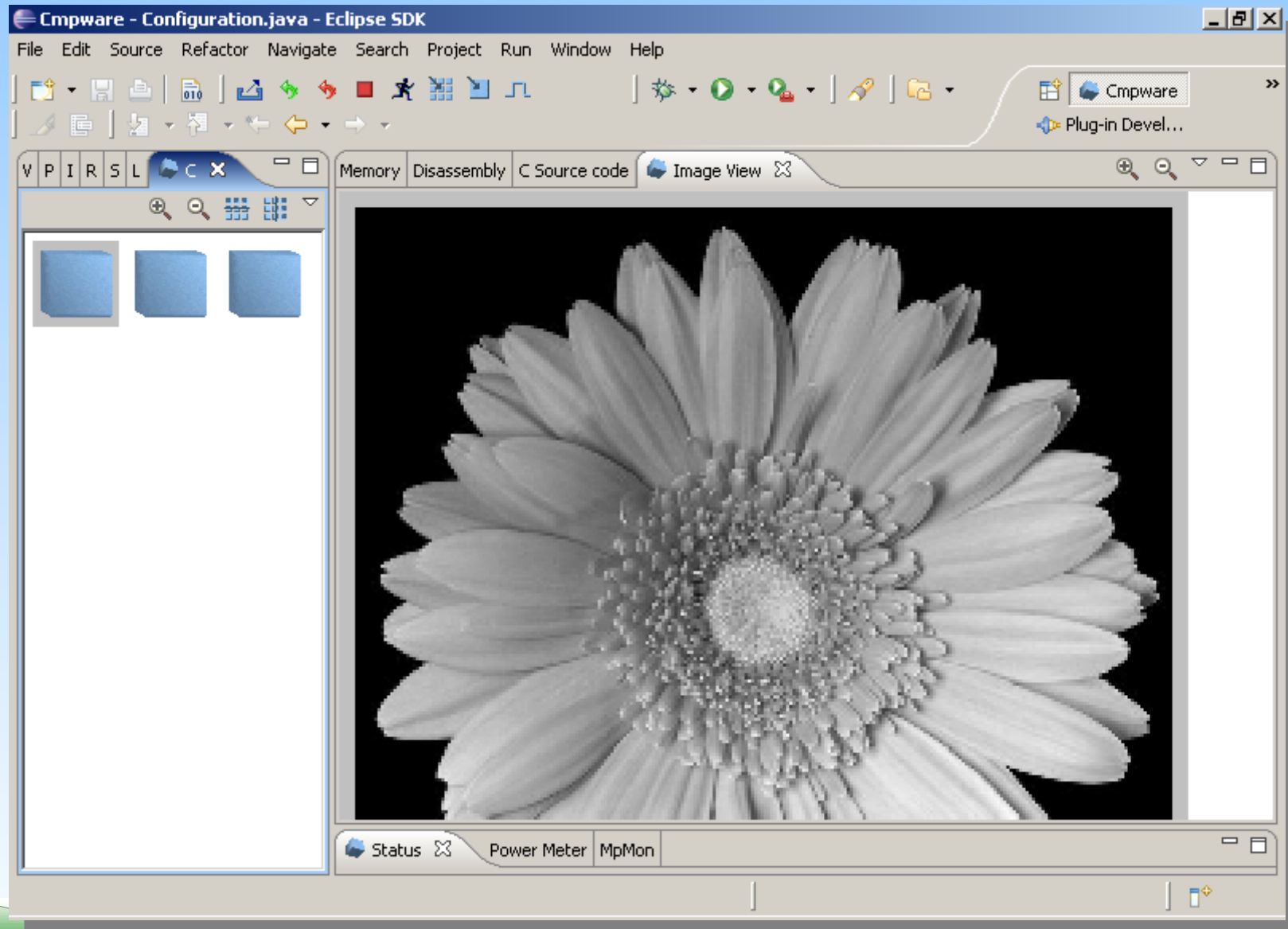


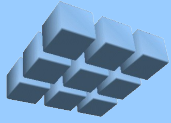
Extra Slides



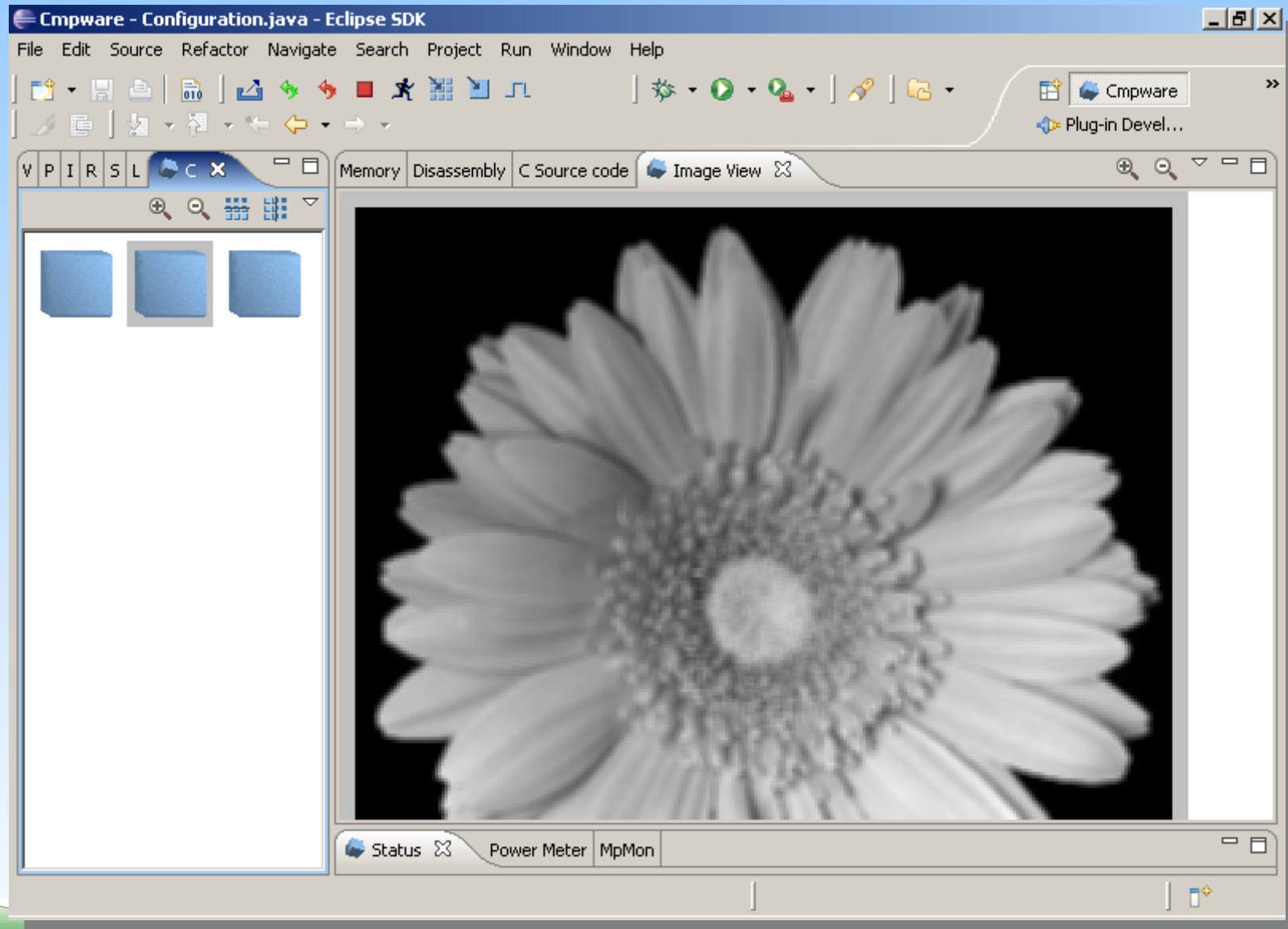


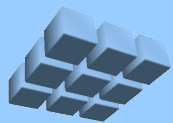
The Original Image





After Smoothing





After Laplacian Edge Detection

