Porting Some Key Caltech & JPL Applications to a PS3 Cluster - A Wild Ride

Paul Springer (JPL), Ed Upchurch (Caltech/JPL), Mark Stalzer (Caltech), Sean Mauch (Caltech), John McCorquodale (Caltech), Jan Lindheim (Caltech), Michael Burl, (JPL)

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

California Institute of Technology
1200 E. California Blvd.
Pasadena, CA 91125

High Performance Embedded Computing (HPEC) Workshop
23-25 September 2008

Theme of Talk: "What Could Possibly Go Wrong?"

- Development Difficulties on a PS3 Cluster
- Some Progress
- Lessons Learned
- Unvarnished view of ongoing work
  - None of the tasks are completed yet
  - Still have unanswered questions
- Plenty of embarrassments--maybe even some in this talk!
  - "Everyone knows the Cell isn't meant to do that"
  - "If you'd just clicked on this link you would have solved your problems"

Introduction

- Last October Caltech's Center for Advanced Computing Research (CACR) purchased 13 PS3's (a lucky number) to build a high performance parallel algorithm testbed for under $10K with a peak potential of a little over 2 TFLOPS single precision.
- The PS3 cluster offers a rich test environment of heterogeneous multi-core nodes with MPI clustering plus the promise of low cost high performance and low power/weight/volume.
- Low cost high performance is attractive for exploring ground based applications such as compute intensive SSA and QMC.
- High performance low power/weight/volume is of interest for space based applications
  - Greater autonomy for deep space missions
  - Downlink data volumes could be significantly reduced

Introduction

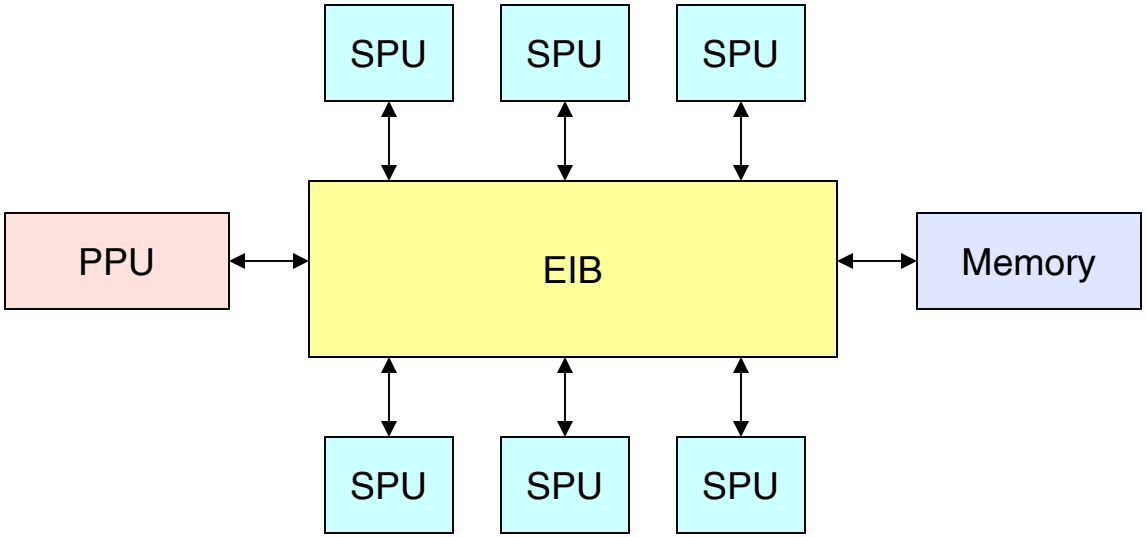
- Our major goal is to assess the actual cost of extracting performance from the relatively inexpensive PS3's. This includes programming time!
- We selected for the first round a set of confirmed "embarrassingly" parallel applications. While not such a challenge in terms of parallelization, the applications selected are of importance to a number of Caltech/JPL users
- Good performance and low porting pain would generate community interest
- Follow on work was planned for more challenging, less parallelizable applications – we have not got that far
- Our budget was $10K for hardware and tools (we got no tools other than free ones) and 1.0 FTE for one year split between three people; No budget for system maintenance – we thought it would not take any

PS3 Cluster Hardware

- 13 PS3 consoles, where each consists of:
  - One 3.2 GHz Power Processing Element (PPE) which has 256 MB of main memory.
  - There is also 256 MB of video memory, which is not available to programmers.
  - One Cell Processor, 6 available SPEs (Synergistic Processing Elements)
    - Each SPE has 256 KB embedded RAM
    - Each SPE running at 3.2 GHz
  - 60 GB disk
  - Blu-ray Disk reader
  - Gigabit ethernet
  - Bluetooth 2.0
  - Wi-Fi network (802.11 b/g)
- 16-port Linksys gigabit switch
- 1 P4 based host machine running Fedora Core 7
- Power Supply and other Hardware Problems
  - Two of our 13 consoles have died; we now leave only 2 on regularly

PS3 Cluster and P4 Host

Cell Block Diagram for PS3



- PPU is PowerPC core
- SPUs are secondary processors
- Only 6 useable SPUs out of 8 total
- One SPU is reserved, one is not accessible

PS3 Software Configuration

- P4 host runs Fedora 7 and SDK 2.1
- All nodes initially installed with YDL 5 and SDK 2.1
  - YDL 5 did not support IBM's SPU-capable Fortran compiler, ppuxlf
- Eventually one node used Fedora 7 and SDK 3.0, a second node YDL 6.0 and SDK 3.0
  - Fedora 7 installation difficulties
    - Fedora Core media not recognized by PS3 BIOS
    - Bootloader had to be downloaded onto pre-configured memory stick
    - Power management needed patched kernel
  - SDK installed easier onto Fedora
    - SDK expects Fedora
    - Some overlap between SDK packages and YDL standard installation
    - Some packages had to be removed to get a consistent system

Versions, Versions, Versions

- **APIs changed on both FFTW and SDK**
- **Version incompatibilities were hard to track**

	Plan	SDK 2	SDK 3	MPI
FFTC	no	yes	yes	no
FFTW 2	yes	no	no	yes
FFTW 3	yes	yes	no	no

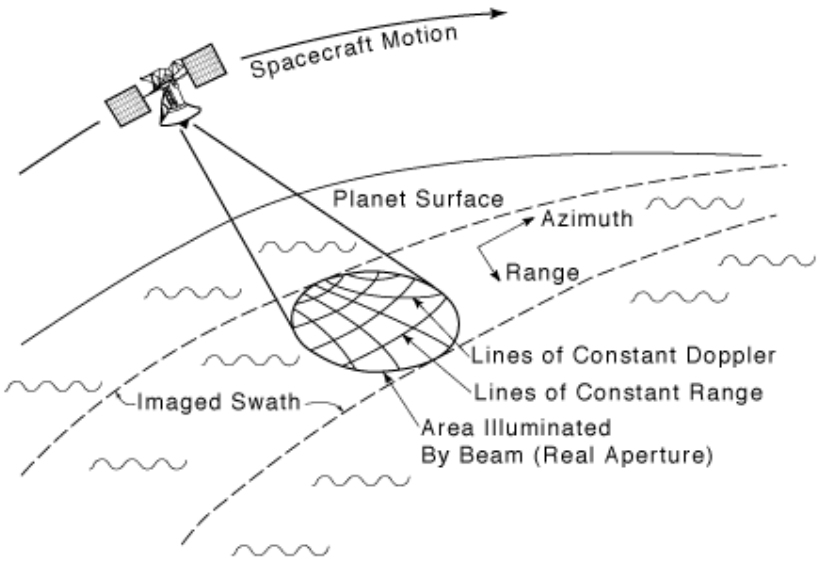
	FFTC	FFTW 2	FFTW 3	precision
SVM	possible	yes	possible	double
ROI	possible	?	yes	single

	SDK 2	SDK 3
Comm code	no	yes

	PPU	SPU	YDL 6	Fedora	SDK 2	SDK 3
gfortran	yes	no	?	?	yes	yes
ppuxlf	yes	yes	?	yes	no	yes

ROI: Introduction

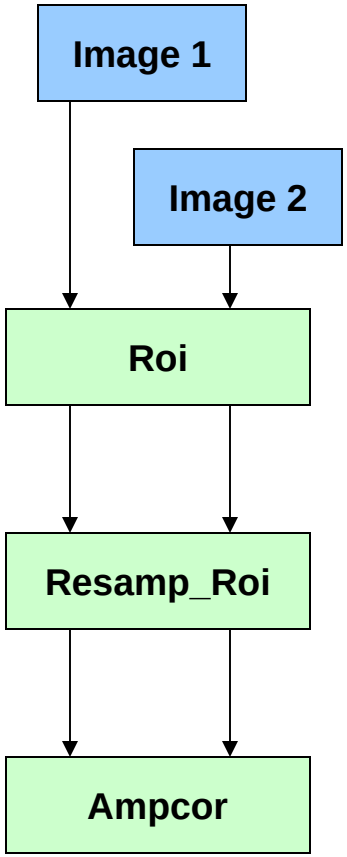
- ROI_PAC, the Repeat Orbit Interferometry Package developed at JPL and Caltech, is a collection of Fortran and C programs bound together with Perl scripts to perform certain routine but important tasks in synthetic aperture radar (SAR) repeat orbit interferometry.
- Individual programs perform everything from raw data conditioning, SAR image processing, interferogram formation, correlation estimation, phase unwrapping, baseline determination, estimation of topography, removal of topography from a deformation interferogram, and geocoding.



- Perl scripts control combining these programs to specifically create a geocoded deformation phase or topographic phase image from two ERS radar images and a digital elevation model, or create a deformation phase image from three radar images without a digital elevation model.
- ROI_PAC has been optimized to reduce programming time not memory used and therefore trades off programming simplicity with use of large (GByte or better) image buffers – for the PS3's we have to optimize minimum use of memory

ROI: Initial Strategy

- Our first port to Cell--and by far the most complicated
- Very large package including scripts and Fortran programs
  - Configuration process searches for known Fortran compilers, does a make for each one, tests each one, and gives results
  - What parts do we port; what parts go onto SPU?
  - Package could handle many tasks; create a single benchmark
- How do we conceptualize the Cell?
  - Seven processors?
  - One processor with 6 accelerators? √
  - Reasoning: SPUs have low memory, no MPI
  - Heterogeneous architecture results in heterogeneous programming model
    - This model necessitated by large differences between PPU and SPU
    - Single model would make for easier development
- Plan: first do MPI port to cluster, then bring in SPU support
  - MPI code had not been used in a while, its status was uncertain
  - Slow parts of code had already been identified, with MPI code added to them



ROI: Approach and Problems

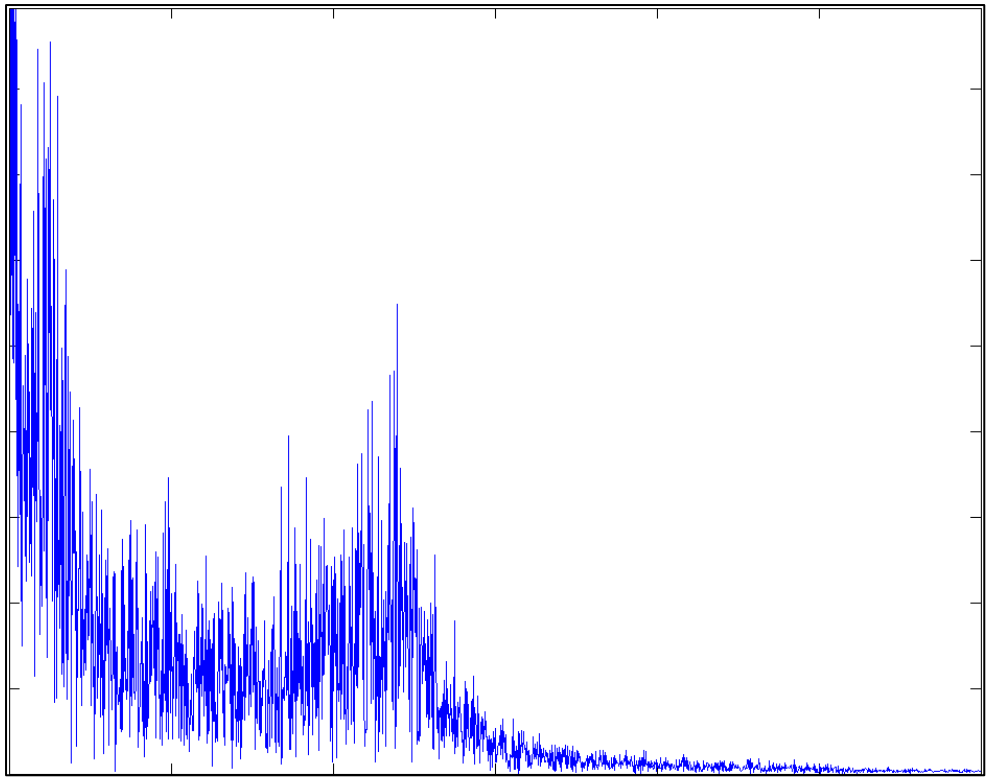
- Built a reference version of roi_pac on the P4 host. Ran package's test script. No problems. (False) confidence builder.
- Built roi_pac on PS3, pointing to ppuxlf, but it crashed in testing
  - Web search revealed that YDL 5 was incompatible
- Generic PS3 Fortran build began running (no SPU support), but then crashed
  - No detailed information, even from gdb
  - One program crashing in the middle of a number that were being executed via test script; very hard to track down
- Two of three main programs in roi_pac needed >1GB memory
  - We built new test script to only exercise smaller program
  - Problems expected for SPU, but not for PPU
- "We don't develop for low-memory environments"

ROI: MPI

- MPI code embedded in the S/W was old and out of date
- Turning it on revealed only minor problems
- Test script limited parallelism to x5
- Code supported parallel file system, but we had none
- I/O was to host's cross-mounted disk
  - Slow
- We chose to benchmark based on processing time, not including I/O time
  - Timing on P4 host: 157 seconds
  - 1 PS3: 294 seconds
  - 5 PS3s (using MPI, but not SPUs): 71 seconds
- Single PS3 run had many page faults, but not the 5-node runs

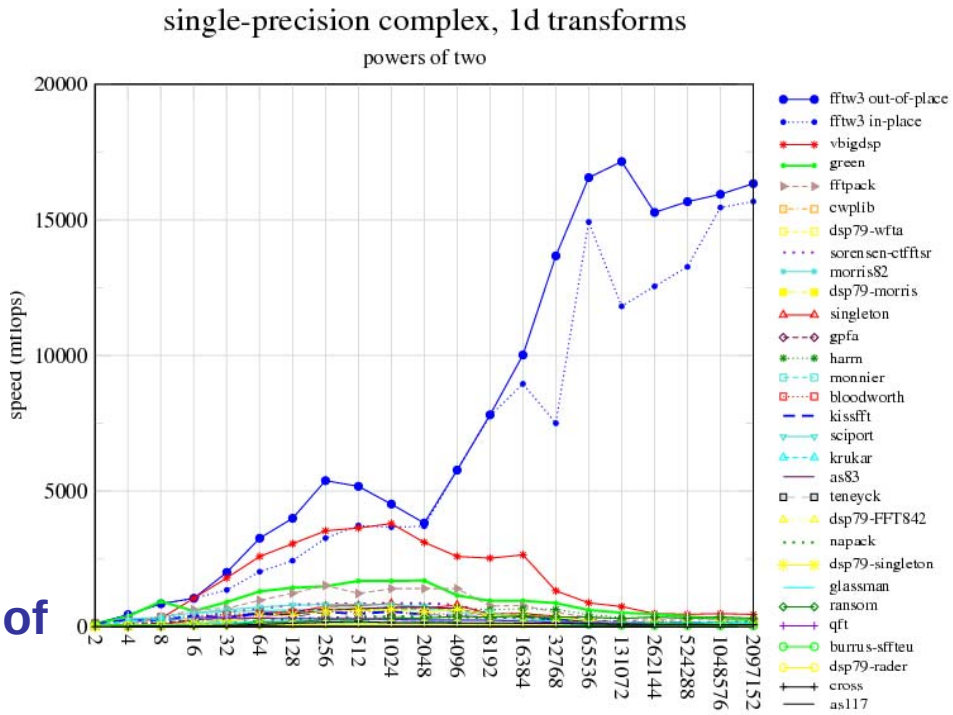
ROI: FFT

- We had heard that FFT performance looked promising on the Cell
  - We looked for applications that made use of FFTs
- Building roi_pac requires an FFT package to be downloaded first
- We chose FFTW as there was built-in support already
- roi_pac README specified FFTW 3--we used 3.1.2
- Statistics showed about 50-70% of time was spent doing FFTs
  - But the run included many page faults, so it bears further investigation



FFTW

- ROI uses FFTW3, SVM uses FFTW 2
- But…only FFTW 3 had Cell support
- FFTW 3.1 had no MPI support (but 3.2 now does)
- FFTW 3.1 required SDK 2
- Our communication code was written using SDK 3
- We eagerly await new versions of software that use SDK 3, and FFTW 3



FFTC

- High speed FFT package from Georgia Tech, customized for Cell
- We wanted to test what performance we could get on PS3, as well as what versions were required
- The released version was written for the SDK 2 I/F
  - We modified it to work with SDK 3
- No 6-node version available
- The 4-node version performed at 5.9 Gflops for 16K complex FFT
  - Lower than we expected; published results showed 22 Gflops on 8 nodes, running on a blade
- No interface for plan generation, like FFTW has
  - Those peak numbers can't be obtained for a real application, unless package is modified

Stochastic Simulation Algorithm (SSA)

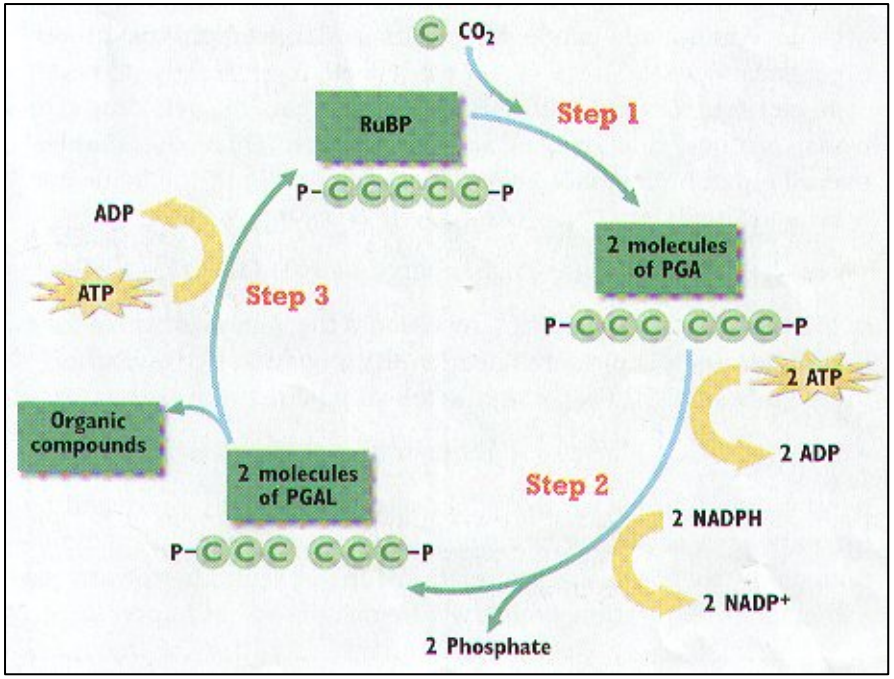
- Algorithm originally due to Gillespie [1976] computes trajectories of continuous-time discrete-state Markov processes
- Stochastic simulation of trajectories of coupled reaction systems with integer species counts (like systems of ODEs with small-integer-valued variables)
- Usually used for rarified-participant chemistry
- Applicable where discreteness of participant cardinality matters
  - biochemical simulation (well-stirred test tube assumptions), aerospace, planetary science, etc.

- Experimental SPU-based implementation of original Gillespie algorithm:
  - Entirely on a single SPU (including model, simulation state and RNGs)

Spatially-Explicit Discrete-State Markov Simulation

- A particle-system simulation of discrete populations of reactants
- Like a spatially-explicit SSA
- Roughly the computation performed by the extremely successful 'MCell' biological simulation tool [Stiles and Bartol 2001]

- Experimental SPU-based implementation:
  - Simple two-species fusion/fission system
  - Everything on one SPU (system, state, RNGs)



Random Number Generation

- Heavy use of RNG for previous two applications--we need good performance
- Improving on IBM SDK Monte Carlo library
  - Uniform Mersenne Twister slower than expected: 6M uint32s/s per SPU
  - We decided to write our own, with GPL license
  - Important to know RNG reliability, for science applications
    - Earlier version of gaussian was not smooth near the gaussian mean
    - Technical report detailing testing of RNG would have saved us work
- Custom uniform Mersenne Twister with period 2^19937 [Matsumoto and Nishimura]
  - 4 x 32-bit uniform unsigned ints per 83 cycles--153M/s per SPU
  - 10k of SPU memory for RNG state
  - Straight C: possibly faster if hand-optimized
  - about as fast per SPU as Virtex-II (FPGA) implementation [Thomas and Luk 2005]
  - Passes “Crush” [L'Ecuyer and Simard 2005]

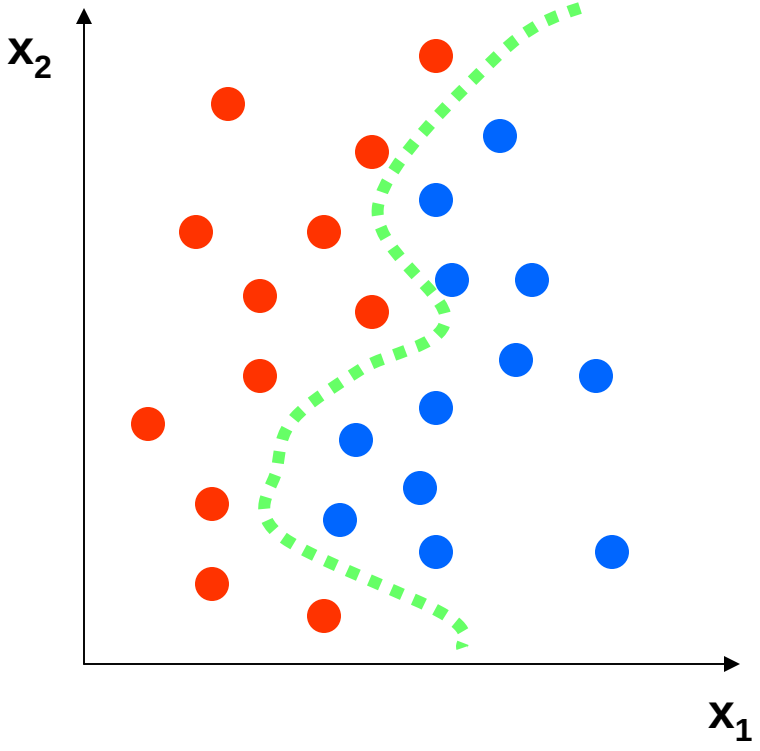
RNG: Gaussian

- Uniform 4 x uint32 -> Gaussian 4 x float
  - [4,4]-rational polynomial approximant of inverse-gaussian-cumulative
    - fit to Chebyshev residual via differential correction [Cheney and Loeb 1961 / Klein 1968]
    - error comparable to SPU float precision
    - future work: SPU error analysis (worst-case ulps), inversion & TestU01
  - piecewise in discrete log of uint32 (count leading zeros)
    - branch-free SIMD in TBD cycles
    - we believe novel approach
  - Same approach can be used to efficiently generate broader class of numbers, such as those needed by the SSA algorithm
- Composite performance: 72 million gaussian-distributed floats/s/SPU

Support Vector Machines (SVM)

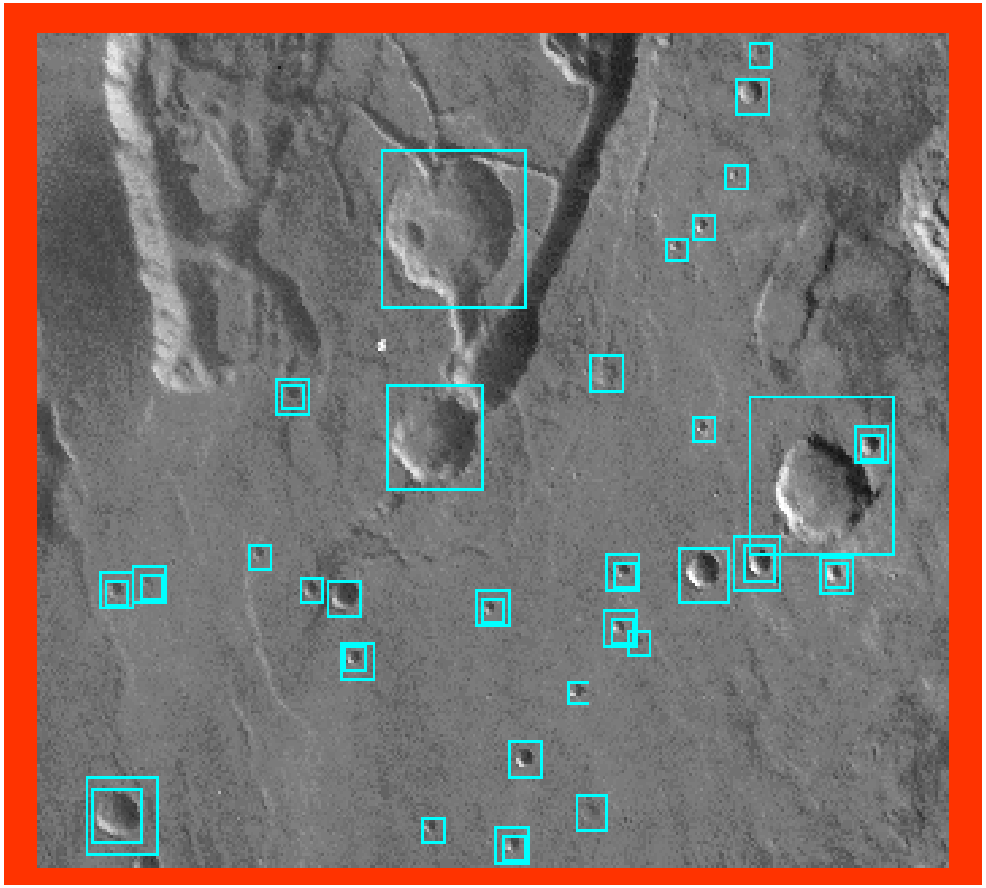
- **Nonlinear classifiers that yield best or close-to-best results across a variety of applications**

- **Sample Applications**
  - **Medical image analysis**
  - **Protein folding, etc.**
  - **Automatic Target Recognition**
  - **Pedestrian detection**
  - **Handwritten Digit Recognition**
  - **Face detection**
  - **Crater detection**



SVM Image Processing Application

- **Brute Force: Compute SVM decision function at every pixel using a sliding window.**
- **Use image pyramid to find objects of different sizes.**
- **Cost of brute force approach *per image pixel* is $O(Mw^2)$ where M is the number of support vectors and w is the width of the sliding window.**
- **FFT and signal proc tricks (e.g., overlap-and-add) can be used to reduce cost per image pixel to: $O(M \log_2(w))$**



Cost still high, but most of the work is to do FFTs.

Port to PS3?

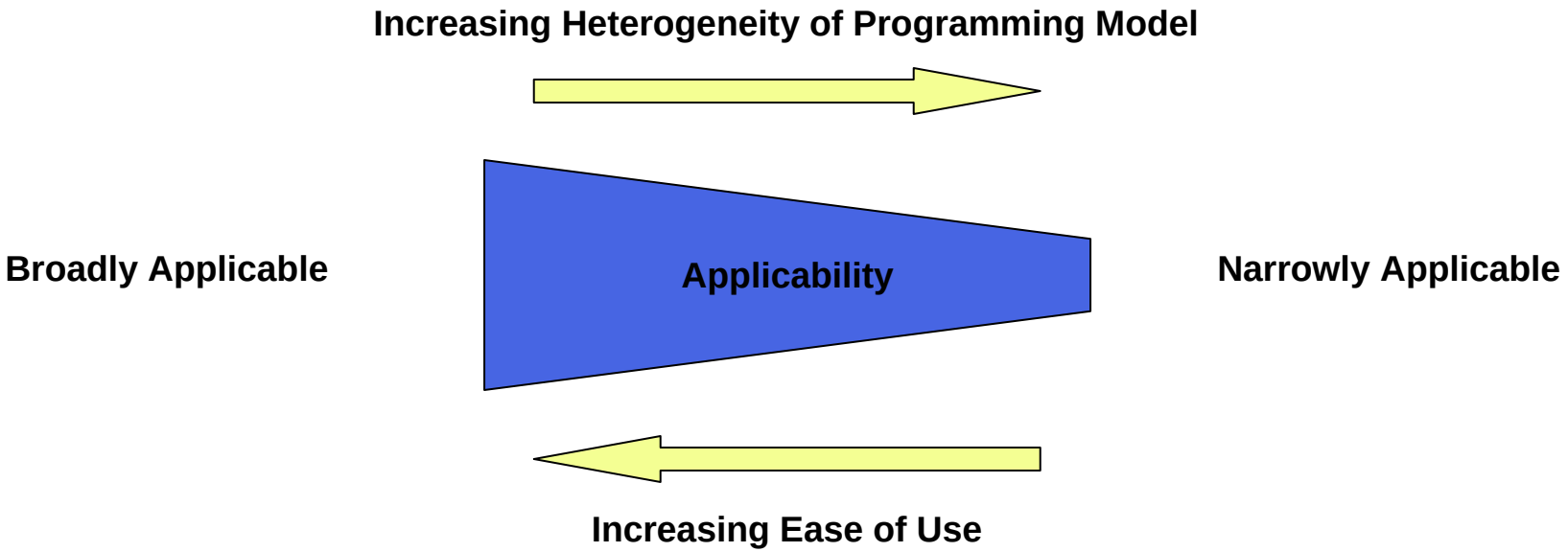
Lessons & Conclusions (1)

- Lower your expectations
  - PS3 is more like 1 processor with 6 accelerators, rather than 7 processors.
- Be selective about which applications to port
  - Experience with Cell is important
  - Better chance of success with new application, than with porting
- Memory considerations
  - 256K for SPU is too small
  - 256M for PPU is limiting
    - Memory limitations can add to cost of porting/developing code
- Language considerations
  - C is best
  - C++ may present difficulties for porting
    - “Hello world” runs out of memory on SPU, using cout()
    - Our C++ objects hid data input inside of object
    - But…efficient DMA usage motivates single DMA operation using a large buffer

Lessons & Conclusions (2)

- Communications
  - Difficult and inconsistent between PPU & SPU
    - Destination addresses must be known at transfer time
    - Different object files imply those addresses must be communicated, not calculated
  - Memory and speed limitations may preclude general purpose messaging layer--a serious problem
- Documentation and Downloads
  - Best site we found is Barcelona Supercomputing Center
    - http://www.bsc.es/plantillaG.php?cat_id=96
    - aggregation of toolchain software in a coherent presentation
  - Wish: IBM SDK libs were delivered in a clean, minimal, distribution independent way (tarballs?) for layering on top of the other software

Tradeoffs with Heterogeneous Multi-Cores



Additional Material

Background--Gedae

- Our combination of software versions on the host (Fedora 7 + SDK 2.1) not fully supported by them
- We tried their Fedora 6 version
- Responsive support, but in the end we could not resolve problems

Issues with Software Installs

- Installing Yellow Dog Linux from the DVD was as simple as installing Linux on a workstation. The media is seen by the PS3 BIOS and boots up into the familiar install menu. For Fedora Core, however, the media was not recognized by the PS3, so a boot-loader had to be downloaded and installed on pre-configured USB memory stick. Also, in order to get power management working with a Fedora Core system, a specially patched kernel has to be installed afterward. We have not tried Fedora Core 9, to see if this has improved.

- IBM seems to use Fedora Core as their standard. The packages for Cell SDK installs typically more easily on Fedora Core, than on YDL. The reason is some overlap in what the standard YDL installation has and what comes with Cell SDK. A number of packages had to be removed from the system, before the installation would succeed.