enParallel, Inc. Desktop Supercomputing Technology

‘Desktop Supercomputing on Graphics Processor Arrays’

J.B. Glenn-Anderson, PhD

CTO ***enParallel, Inc.***

10,000-foot View - I

- ***ePX Framework*** leverages GPU technology for hi-performance scientific computing
  - PC form-factor, standard OS/Compiler technology
  - True supercomputer performance/scaling properties
    - 'Scatter-Gather' vs. 'Co-Processor'
  - Accelerates complete applications
    - Fundamental *dynamic-dataflow* application design representation,
    - Cluster/CPU/GPU Array (GPA) processing pipeline at generalized process queues.
  - Composite SIMD/SIMT (GPU), SMP (multi-core CPU), and Distributed (CLUSTER) processing model
    - Applied to *dataflow-object/component-task/algorithmic-kernel* design representation hierarchy.

10,000-foot View - II

- ePX Framework - Advantages:
  - Performance
    - Single GPU can deliver up to 500x acceleration (e.g. Monte Carlo, ‘stenciled’ linear [FDTD]).
  - Scalability
    - CPU/GPA pipelining hides CPU processes,
    - Extended tripartite linear-scaling ‘$N_{NODE} \times N_{GPU} \times N_{TP/GPU}$’ across Cluster/CPU/GPA resources
  - Flexibility
    - Applicable to diverse algorithmic kernels, complex dataflow structures, (re: SIMT pipeline ‘reuse’),
    - Abstracted Distributed/SMP/SIMT processing model,
    - Portable/Reusable software.

10,000-foot View - III

- ***'ePX Framework'*** – How it works
  - Dynamic-Dataflow design representation
    - Originating node elaborates an initial dataflow object
    - Dataflow object is parsed into component tasks processed locally or scattered onto Cluster according to a ***distributed-recursive*** schema, (i.e. identical component-task processing at every node).
    - Non-Blocking (asynchronous) API calls are employed to overlap Cluster, CPU, and GPA processing.
  - Process-Scheduler
    - Structures all scatter-gather operations and process synchronization,
    - Algorithmic kernels are extracted from component tasks and mapped to local CPU/GPU resources
  - Hardware details are pushed to service methods on generalized process queues:
    - Cluster - multicore CPU - GPA

Graphics Acceleration is Driving PC Evolution..

- Overall evolutionary trend whereby PCs supplant workstations in complex applications
  - Fundamentally a *Moore's Law* phenomenon
- Ever increasing demand for graphics performance
  - AutoCAD, PhotoShop, Gaming, et al.
- PC Architectural Evolution
  - PCIe 2.0 x16
  - CPU/MCH integration, (re: *'NorthBridge'*)
  - Direct MCP connection to CPU, (re: *'SouthBridge'*)

CPU/GPU System Architecture..

- Trend: PCIe 2.0 x16 supplants AGP for high performance graphics
  - PCIe 1.1 x16 offers 4GB/s bidirectional transfer rate,
  - PCIe 2.0 doubles to 8GB/s.
- Trend: GPUs evolve as generic vector processors
  - Open APIs enable 3rd party application development.
- ‘Ganged’ GPUs on multi-PCIe slot motherboards, (re: NVIDIA SLI), not optimal for complex scientific applications:
  - CPU/Coprocessor use-model depends upon execution of single algorithmic kernel for extended periods.

GPU Advantages..

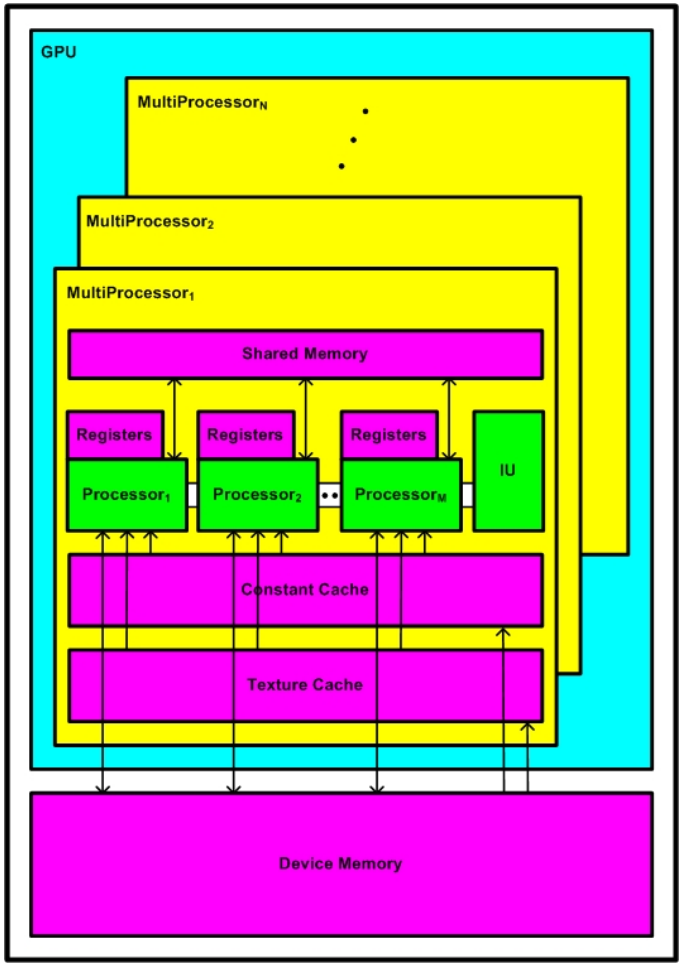
- Hardware abstraction at API, standard software development flow and design representation.
- Multithread Programming Model
  - Avoid FPGA HDL-bottleneck, development-flow complexity.
- Parallel Acceleration
  - Single Instruction Multiple Thread (SIMT) processing model equivalent to powerful vector processor.
- Cache parallelization
  - Addresses fundamental thread-synchronization problem,
  - Enables hi-RADIX concurrent memory access
- Floating-Point Processing,
  - Avoids complexity and specificity of fixed-point implementations,
  - Obviates associated dynamic-range and loss-of-precision issues,
  - Double-Precision available.
- Pre-existing Body of Algorithmic I/P,
  - Fast-Track to Scientific-Computing, Image/Signal-Processing market penetration.

GPU Hardware Architecture I

- Single Instruction Multiple Data/Thread (SIMD/SIMT)
  - Cyclostatic multi-Processor
- Parallel Cache
  - Shared Memory +Constant + Texture Cache
- Parallel (‘$O(10^2)$’) Thread Processors
  - Hierarchical memory architecture organized CPU $\leftrightarrow$ Device Memory $\leftrightarrow$ Shared Memory
  - Separate Register Bank for each TP
  - Texture-Memory optimizes on spatial-temporal coherence

GPU Hardware Architecture II

Example GPU logical resource organization



Desktop SuperComputer (DSC) Processing Model I

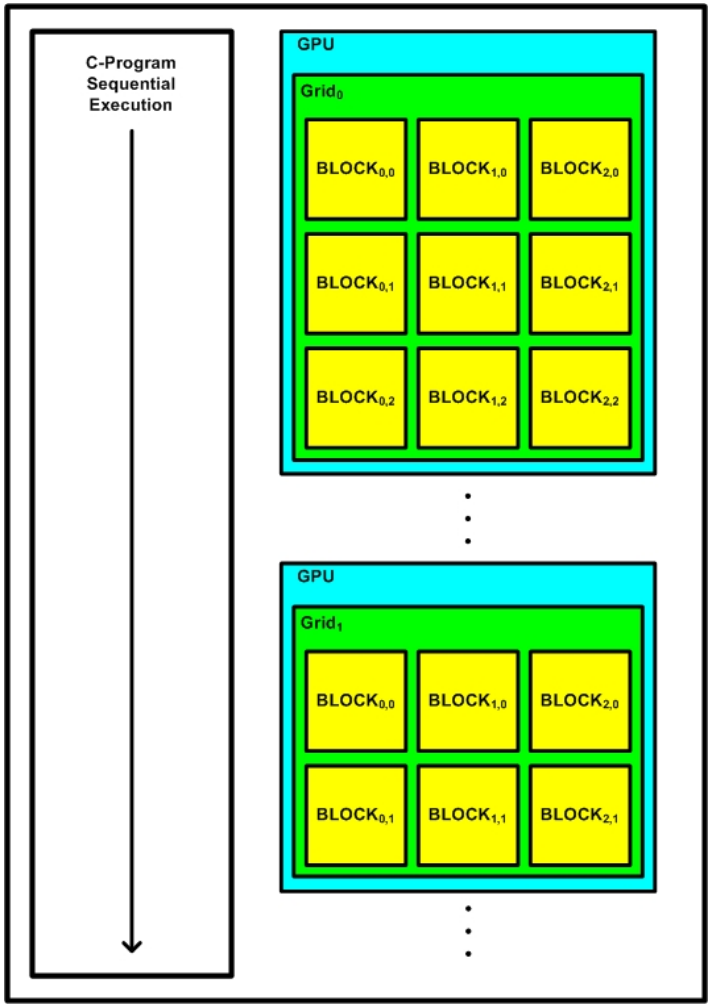
- Why do we need it?
  - With consideration of complex applications, a fundamental performance issue emerges in connection with instruction-pipeline ***cyclostatic residency***,
  - CPU/GPU-Coprocessor model is intended for single algorithmic kernel executing over long interval,
  - ***Scatter-Gather*** on GPU Array enables; (1) efficient SIMD/SIMT acceleration over diverse algorithmic kernels, and (2) full concurrency over all GPU instances.
- 4x fundamental ePX DSC principles..
  - Supercomputing-styled scatter-gather,
  - CPU/GPU process pipelining (scheduler),
  - Apply hierarchical *coarse-grained*-to-*fine-grained* parallelism based upon synergy of distributed, SMP, and SIMD/SIMT processing models,
  - Reuse GPU instruction-pipelines part and parcel of algorithmic-kernel scheduling.

DSC Processing Model II

- GPU Applications Programming Interface (API)
  - Virtualizes GPU hardware in form of standard C/C^{++} function calls,
  - No requirement for specialized parallelizing compiler/OS runtime support
- API implements streaming communications model
  - MTP instruction pipeline initialization
  - GPU WRITE/READ transaction buffer
- ***'Non-Blocking'*** (asynchronous) GPU memcopies at CPU
  - Enables CPU/GPU process pipelining,
  - Enables GPA scatter-gather and GPU process pipelining,
  - Enables GPA load-balancing, and (GPU) instruction pipeline reuse
- PC-based Supercomputing
  - Leverage existing Symmetric Multi-Processing (SMP) capability available on modern multicore CPUs,
  - Fully compatible with standard PC applications environment.
    - MSVS/GCC + Windows/Linux

Desktop SuperComputer Processing Model III

Concurrent CPU/GPU Execution, (i.e. ‘process-pipelining’).



DSC Processing Model IV.. (Theory)

Assume a form of Amdahl's Law specialized for CPU/GPU pipelining $A = \dfrac{1}{\left(1 - C_{CPU}P_{CPU} - P_{GPU}\right) + \dfrac{P_{GPU}}{N}}$

$C_{CPU} \equiv$ pipeline efficiency constant $\in [0,1]$, $P_{CPU} \equiv$ pipelined CPU code fraction, $P_{GPU} \equiv$ GPU accelerated code fraction.
Of particular interest is the limiting case:

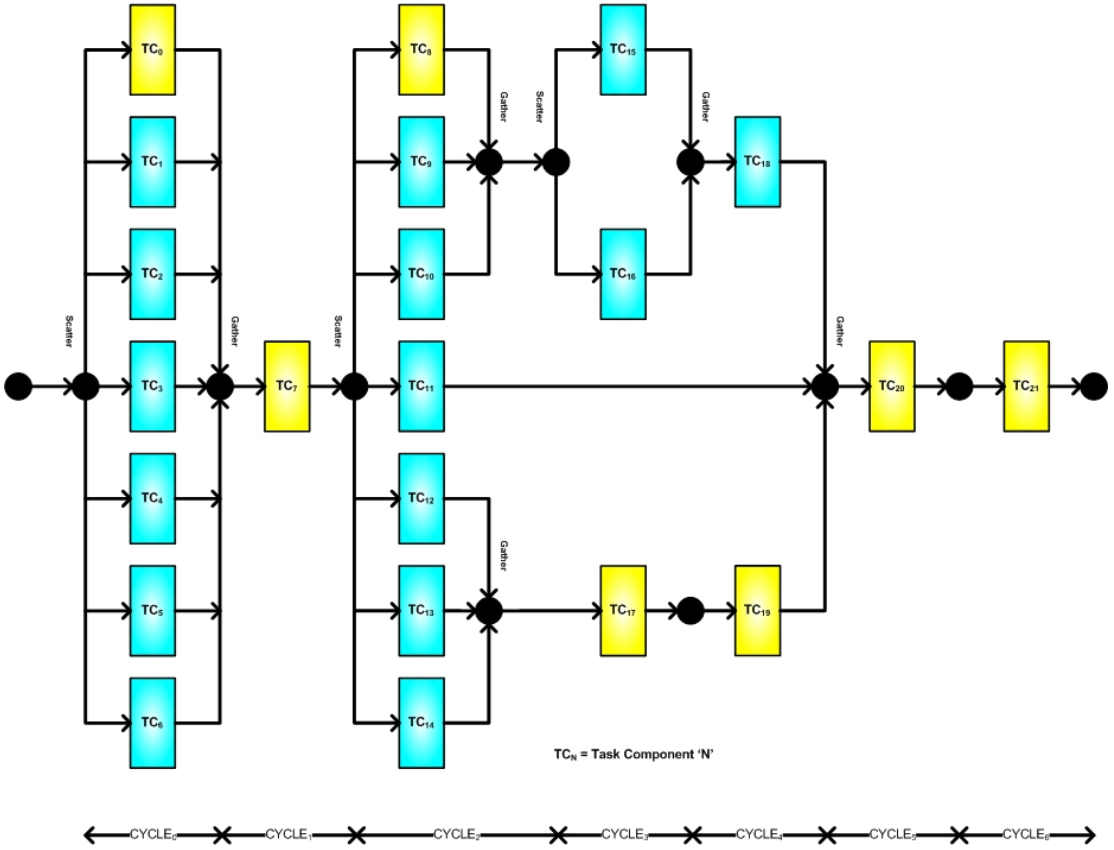
$$\lim_{N \to \infty} \left(\frac{1}{\left(1 - C_{CPU}P_{CPU} - P_{GPU}\right) + \dfrac{P_{GPU}}{N}} \right) = \frac{1}{1 - C_{CPU}P_{CPU} - P_{GPU}}$$

However, with $C_{CPU}P_{CPU}$ sufficiently large, $\dfrac{P_{GPU}}{N} >> \left(1 - C_{CPU}P_{CPU} - P_{GPU}\right)$. Amdahl's Law then becomes:

$$A \cong \frac{1}{\dfrac{P_{GPU}}{N}} = \frac{N}{P_{GPU}} \cong N \,;\, A_{CPU-GPA} \cong N_{GPU} \cdot N_{TP/GPU} = N_{GPA}|$$

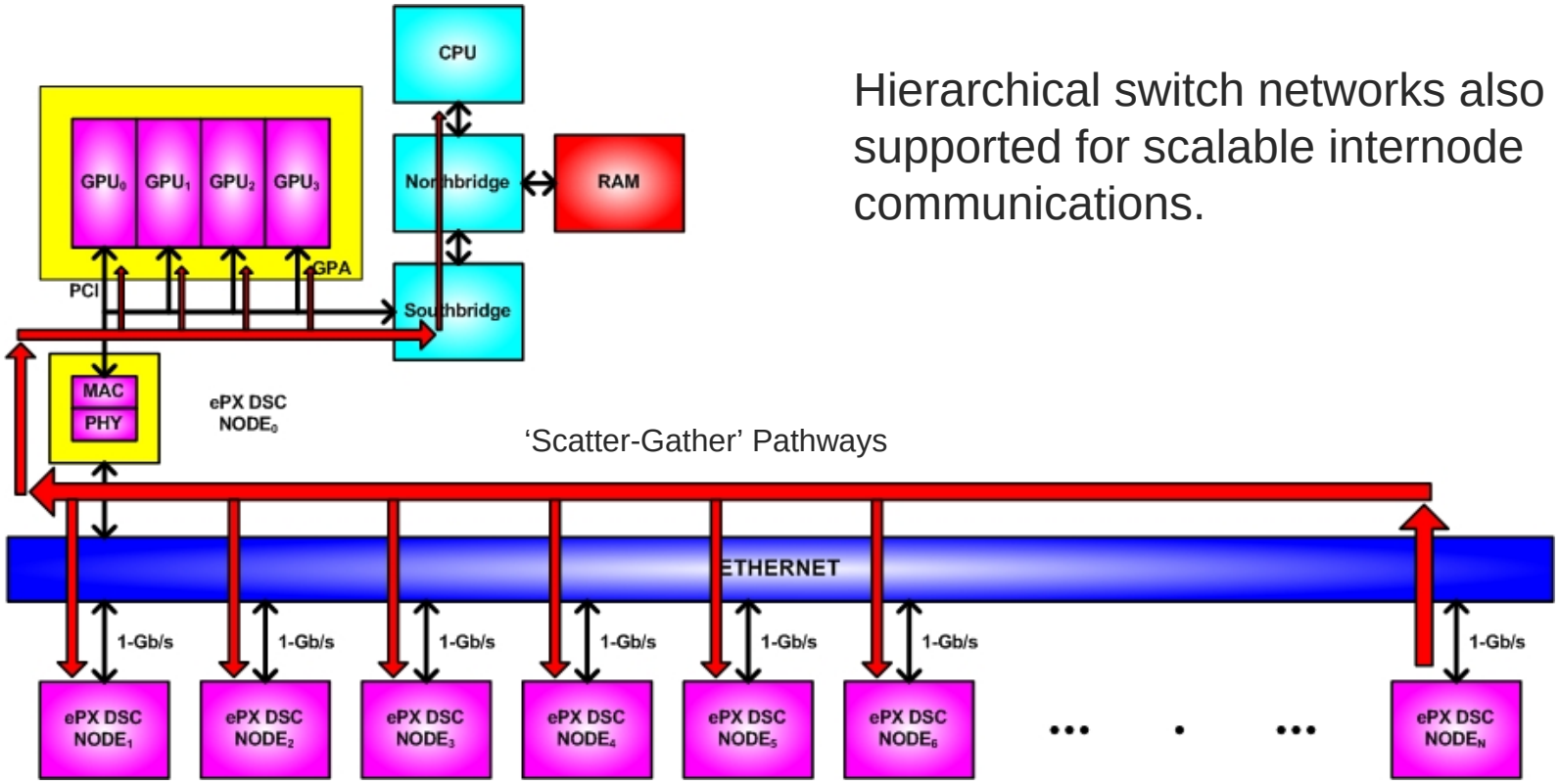
$$A_{MCPU-GPA} = \frac{1}{\left(1 - C_{MCPU}P_{MCPU} - P_{GPA}\right) + \dfrac{P_{GPA}}{N_{GPA}}} \,;\, C_{MCPU} > C_{CPU} \rightarrow A_{MCPU-GPU} > A_{CPU-GPU}$$

Dynamic-Dataflow Design Representation..



Example ‘dataflow-object’

Cluster Architecture..

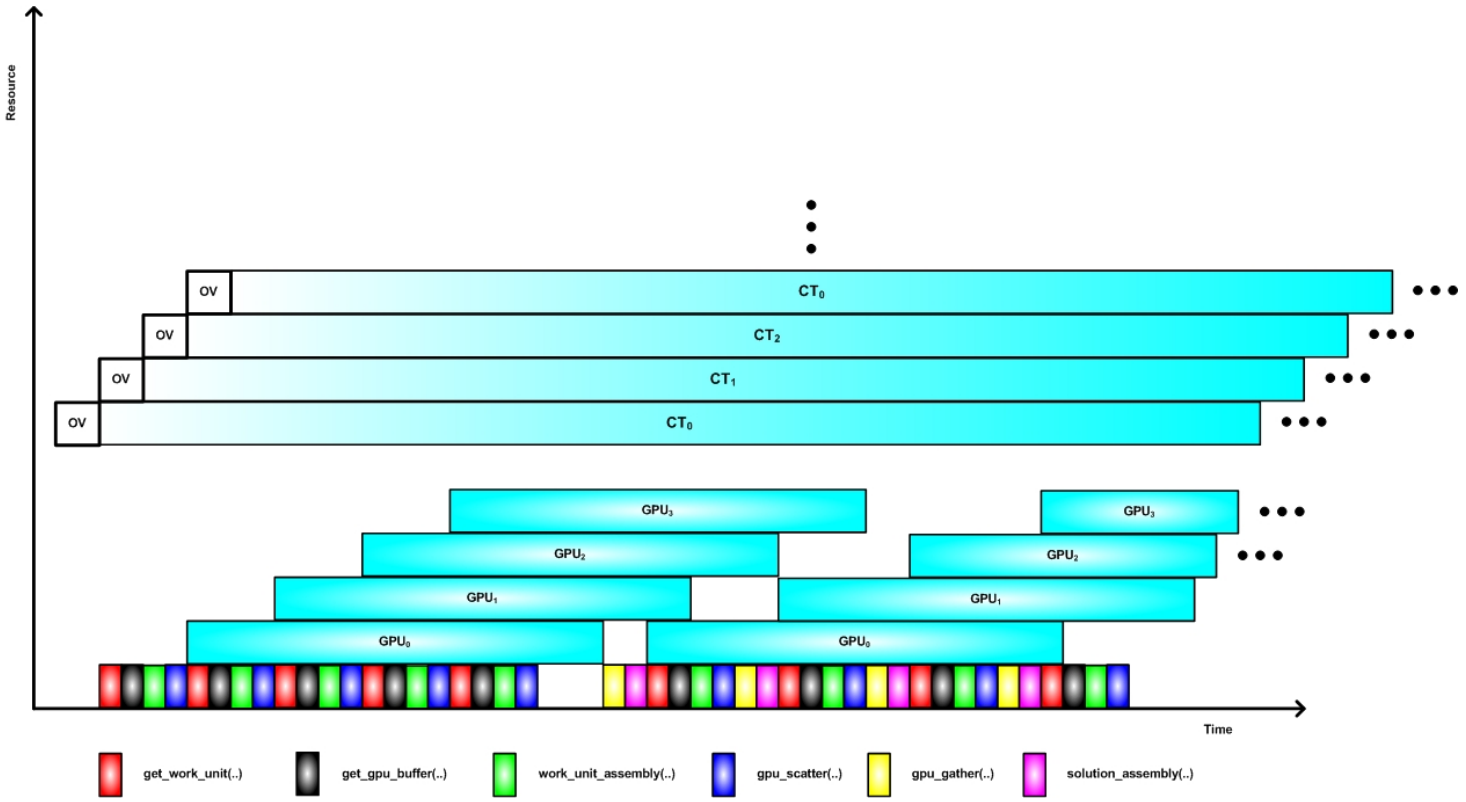


Hierarchical switch networks also supported for scalable internode communications.

Cluster Processing Model..

- Overarching ***distributed-recursive*** cluster processing model (‘divide-and-conquer’).
- Dynamic-Dataflow design representation is incrementally elaborated at originating node and propagated throughout cluster in form of ***dataflow-objects***.
- Any node parses a received dataflow-object into ***component-tasks***; locally processed component-tasks are further parsed into ***algorithmic kernels*** and applied to multi-core CPU/GPA process queues.
- Remaining component tasks are distributed onto unused remote cluster resources.
- Datapath may be optionally virtualized based upon data-server transactions at the originating node.

Cluster/multicore-CPU/GPA Composite Process Schedule..



GPA-based Benchmark Projections

Nominal Experimental Platform: ~2.3GHz dual-core (Intel) processor featuring 2GB RAM, WinXP OS, and a GPA consisting of 4x NVIDIA GeForce 8800GTX graphics cards, (i.e. 128 thread processors per GPU).

Algorithmic Kernel	Acceleration
Computational Chemistry: 2nd order Moeller-Plesset	x17
Life Sciences: Smith-Waterman	x120
Finite Difference: Heat Equation, SOR (Gauss-Seidel solver)	x68
FEM multi-grid: Mixed Precision Linear Solvers	x108
Image Processing: Optical Flow	x220
Image Processing: Cubic B-spline interpolation on 3D textures	x1308
CFD: 3D Euler solver	x116
CFD: Navier-Stokes (Lattice Boltzmann)	x400
Signal Processing: Sparse Signal Recovery from Random Projections (NP-hard combinatorial optimization)	x124
Computational Finance: Quantitative Risk Analysis and Algorithmic Trading	x200
Computational Finance: Monte Carlo Pricing	x320
EDA: static timing analysis	x1040
EDA: SPICE simulator	x32

Prototype Test Platforms…

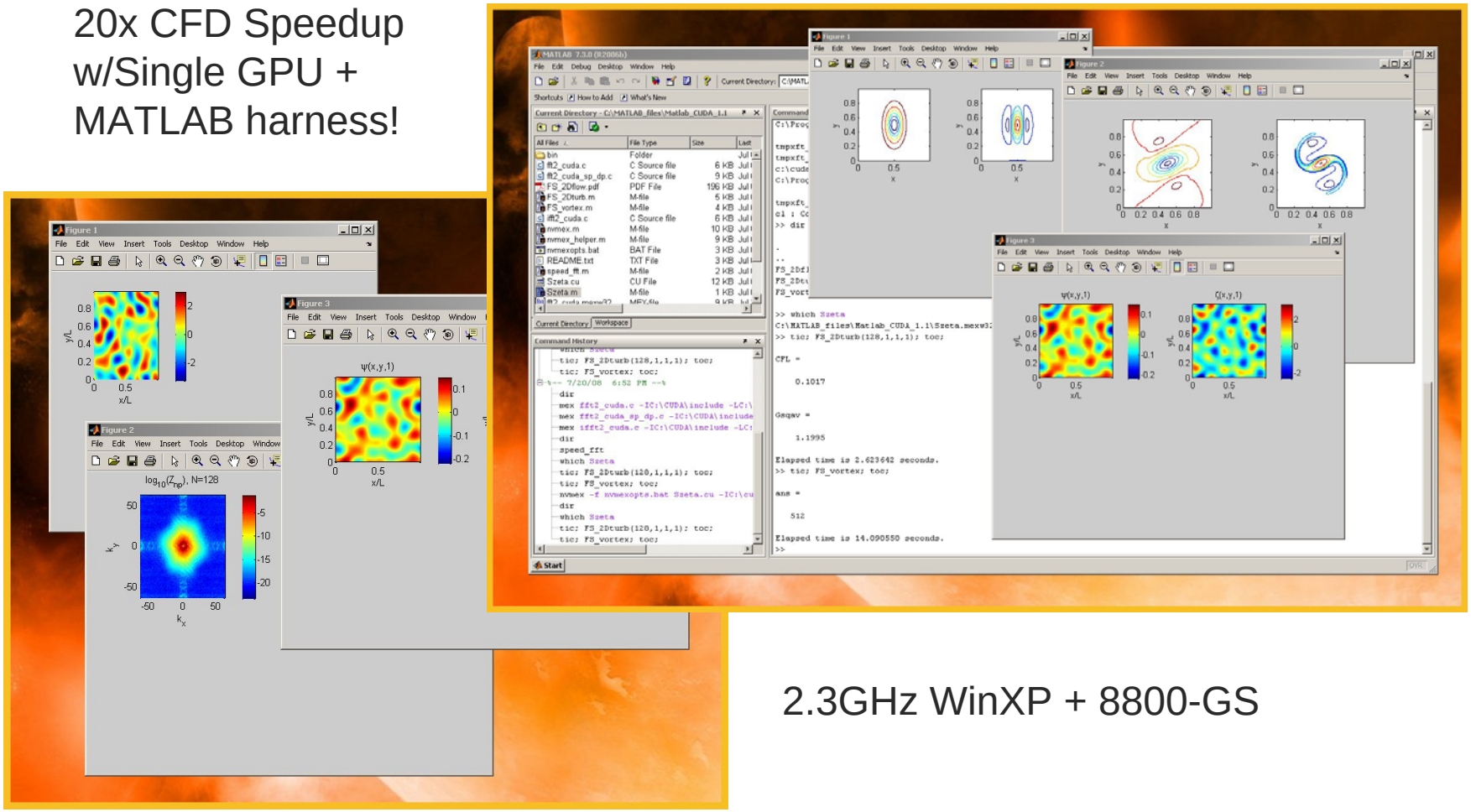
- Platform:
  - ATX form-factor, abit KN9-SLI motherboard, AMD A64 x2 4400+ (2.3GHz) CPU, 2GB 800MHz DDR2, 2GHz HyperTransport system bus,
  - 2xPCIe x16 slots, 600W P/S,
  - WinXP Professional.
- GPU:
  - **I** - 2x eVGA GeForce 8800 GS (1xGPU), 96x stream processors, 384MB memory, 192b memory interface, 38.4Gb/s memory bandwidth,
  - **II** - 2x eVGA GeForce 9800 GX-2 (2xGPU), 256x stream processors, 1-GB memory, 512b memory interface, 128Gb/s memory bandwidth,
  - Operated in non-SLI mode, (1xGPU interleaved with display).
- Software:
  - MATLAB 7.3 (2006b),
  - MicroSoft Visual Studio C++ 2005 Express,
  - NVIDIA CUDA + SDK + Profiler.

Benchmark Test (I)…

- Fourier pseudospectral simulation – 2D fluid dynamics (C. Bretherton, Univ. Washington)
  - GPU-based application is virtualized inform of standard MATLAB function-call, based upon ‘MATLAB EXternal’ (MEX) API,
  - MEX-wrapper $\rightarrow$ CUDA (thread-scheduled ‘C’) $\rightarrow$ Compile + Link against MATLAB + CUDA libraries $\rightarrow$ DLL; access MicroSoft Visual C^{++} runtime libraries.
  - Algorithm is ‘2D FFT-Intensive’, (i.e. uses optimized NVIDIA CUFFT libraries),
  - Extensible to arbitrary coarse/fine-grained MATLAB model components.

Benchmark Test Results (I)…

20x CFD Speedup w/Single GPU + MATLAB harness!



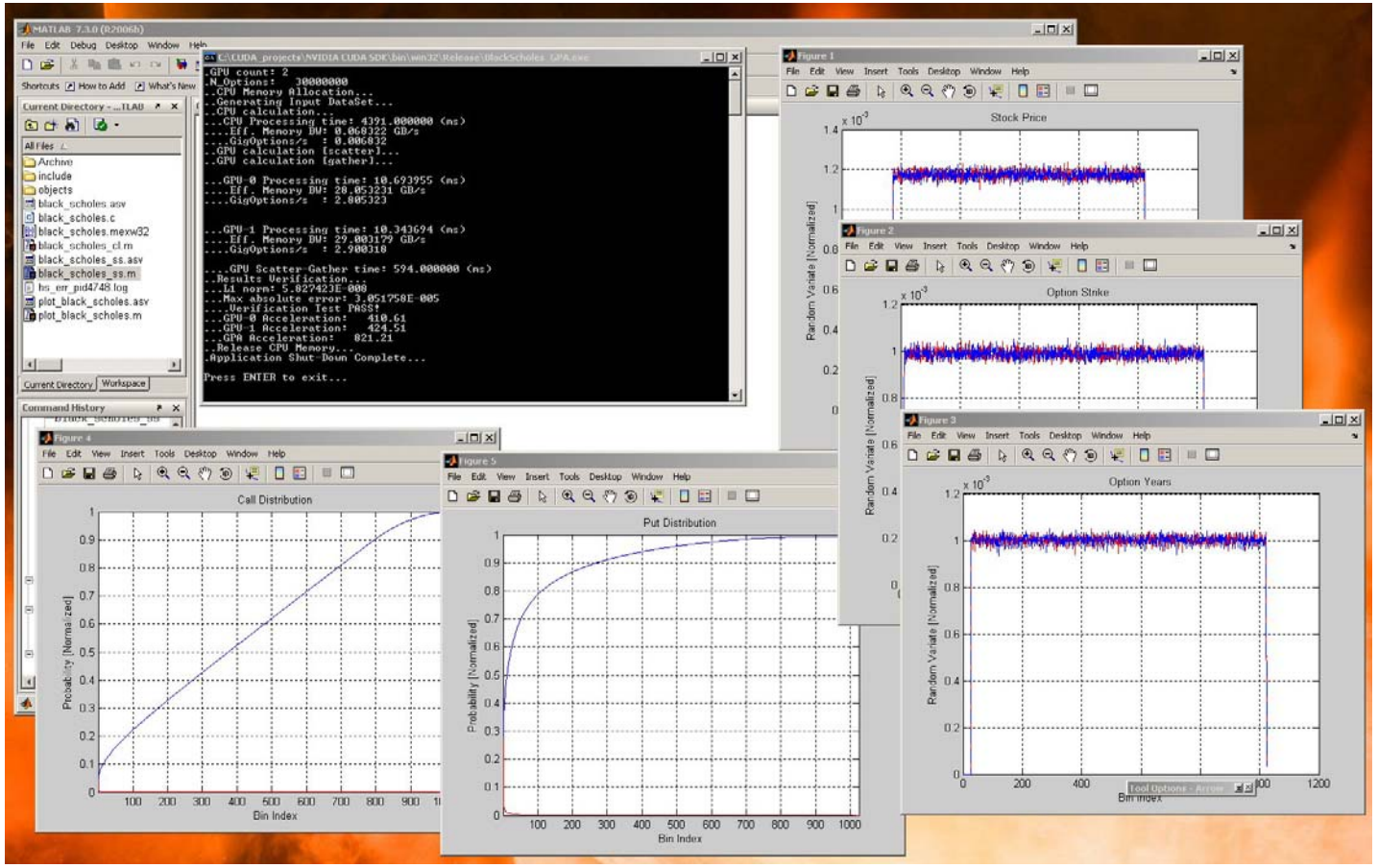
2.3GHz WinXP + 8800-GS

Benchmark Test (II)…

- Black-Scholes Monte Carlo Simulation..
  - GPA-based application is virtualized inform of standard MATLAB function-call, based upon ‘MATLAB EXternal’ (MEX) API,
  - MEX-wrapper $\rightarrow$ CUDA (thread-scheduled ‘C’) $\rightarrow$ Compile + Link against MATLAB + CUDA libraries $\rightarrow$ DLL; access MicroSoft Visual C^{++} runtime libraries.
  - ePX *scatter-gather* distributes random variate processing to GPA elements based upon detection of ‘N_{GPU}’ GPUs.
  - Simulation results are ‘binned’ in post-processing step and passed back to MATLAB environment, (i.e. for plot generation, or subsequent simulation processing),
  - Conceptual basis for MATLAB as generic application delivery platform.
    - Extensible to **‘SciLab’**, **‘RLab’**, **‘SOFA’**, and **‘Octave’** applications.

Benchmark Test Results (II)

Black-Scholes: $3x10^7$ Options - 2.3GHz WinXP Dual 8800-GS - **821x Acceleration!**



Summary..

- **enParallel (ePX) DSC is really a new class of supercomputer!**
  - Based upon CPU/GPA architectural template,
  - Synergizes with SMP (multicore) and distributed (cluster) processing models,
  - Builds upon traditional supercomputing technique:
    - CPU/GPA process pipelining, scatter-gather, hierarchical (coarse/fine-grained) parallelization.
- Excellent GPA/Cluster scaling properties.
- The focus is acceleration of complex scientific applications incorporating diverse algorithmic kernels.
  - Distinct from standard CPU/GPU-Coprocessor model,
  - Enables efficient parallel-processing over diverse algorithmic kernels.
- Major advantages over competing acceleration technologies..
  - FPGA-based accelerators,
    - Essentially an algorithmic ‘point’ solution, (re: complex and relatively unavailable partial reconfiguration technology, and implicit architectural specificity).
    - Problematic and complex development flow,
    - Nominally rudimentary memory model, (i.e. significantly limits processing model options).
  - multicore CPU (SMP),
    - GPU exhibits; (1) significantly higher parallelism, and (2) superior performance when applied to kernels for which the SIMD/SIMT processing model is optimal.