Computer Systems, Inc.
MERCURY

Partitioning Computational Tasks within an FPGA + RISC Heterogeneous Multicomputer

John Bloomfield, Mercury Computer Systems, Inc.

HPEC – September 2002

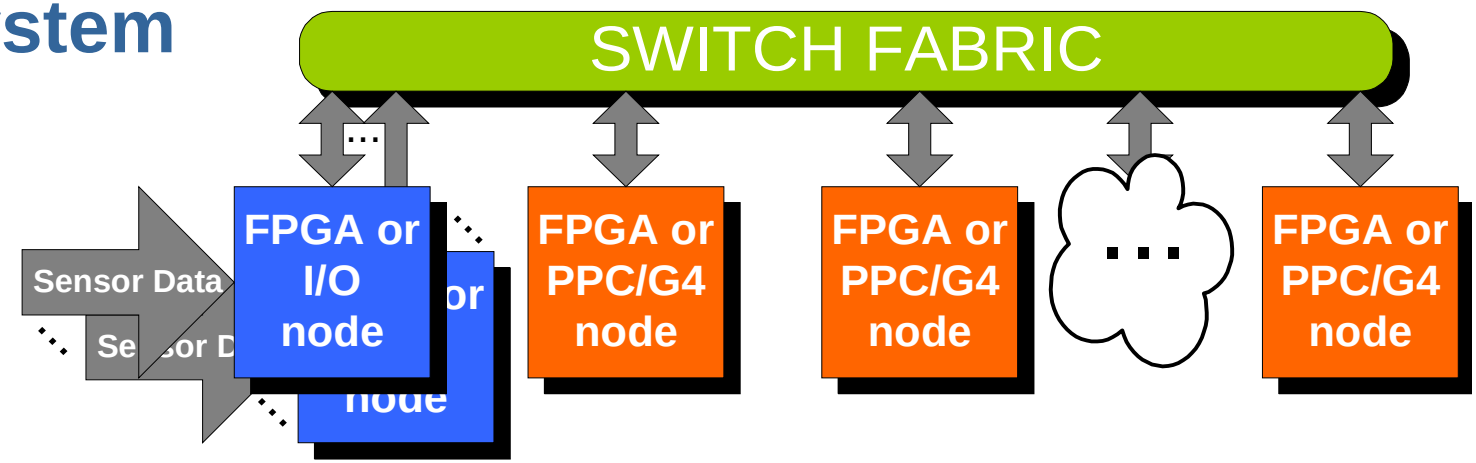
The Ultimate Performance Machine

Agenda

- Why worry about partitioning?
- How we partitioned a real-world problem and the benefits we got.
- Generalization of partitioning concepts and factors.
- Concluding remarks.

Why Partition?

- You are developing an embedded system
  - Constrained by performance, size, weight, cost, & power
  - You can't meet the constraints with RISC processors only
  - But FPGAs can outperform RISC processors by a factor of 10 or more on ***some*** tasks
- Therefore, you require a heterogeneous system

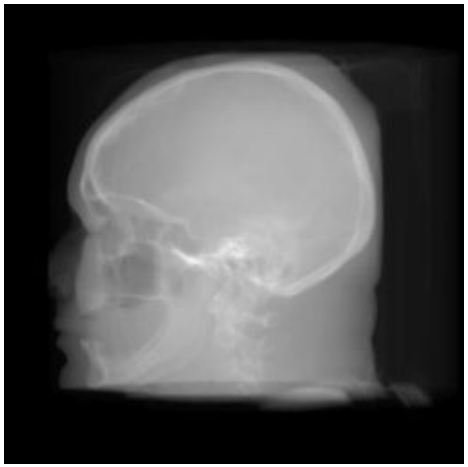


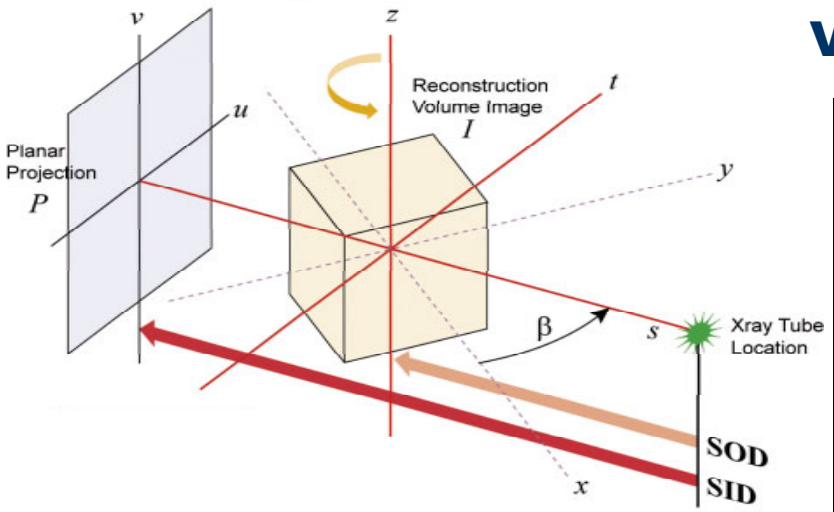
- This inevitably leads to the question
  - "How do I partition my computational tasks between all the heterogeneous processors I have at my disposal?"

Real-World Example

- We selected a problem that was difficult to solve efficiently with a G4 PowerPC:

Sample 512^2 Projection Frame

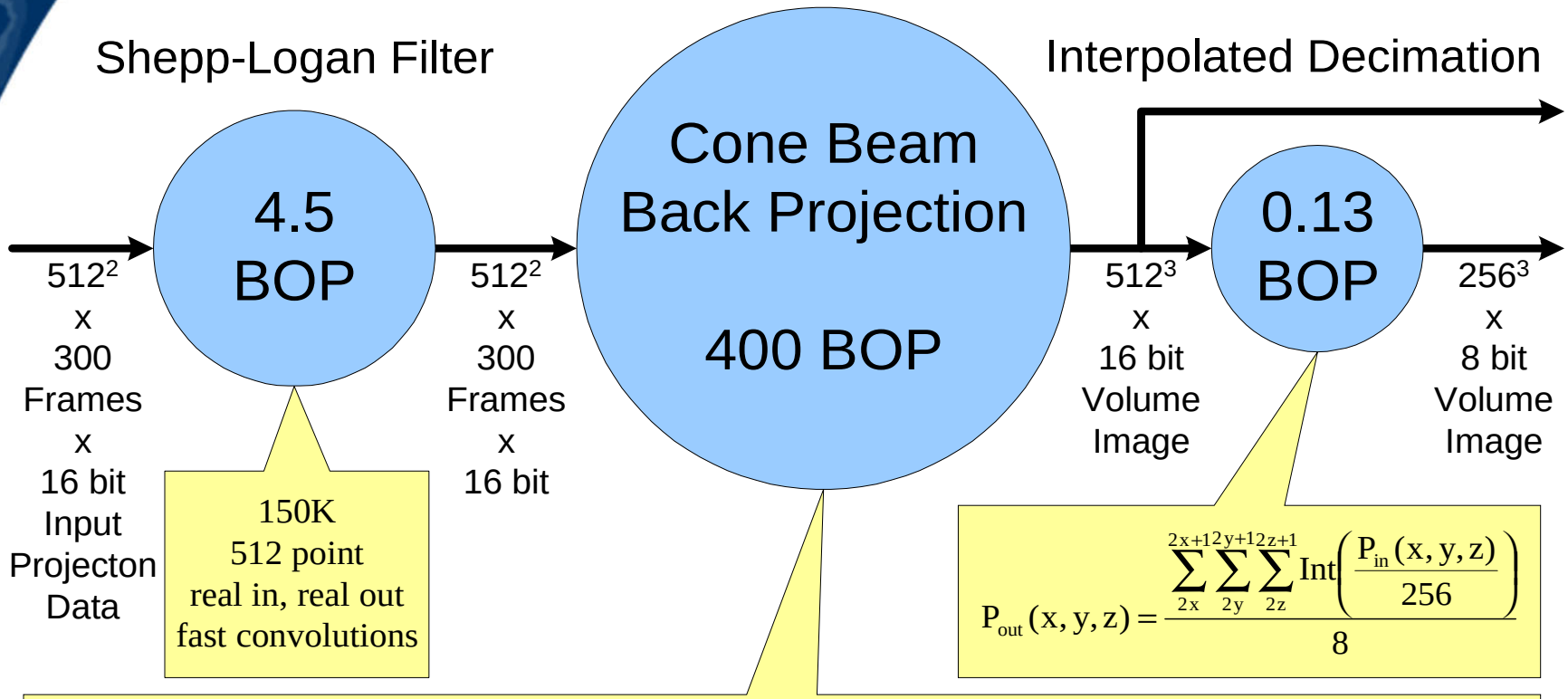




Sample 512^3 Volume Renderings

- We took that 3-D volume reconstruction problem and partitioned it between RISC and FPGA nodes.
- We then implemented it, demonstrated it, and reported the results.

The Problem Tasks



$$I[x,y,z] = \sum_{\beta} P[u(x,y,\beta), v(x,y,\beta), \beta] \cdot w(x,y,\beta)$$

I = 3-Dimensional output image array with x, y, and z-axes
P = 2-Dimensional filtered projection data with u and v axes at all projection angles (β)
β = Projection angle
S = Projected image voxel location onto the s-axis
T = Projected image voxel location onto the t-axis
w = Weighting factor based on source distance to image voxel location
SID = Source-to-image (detector) distance
SOD = Source-to-object (center of rotation) distance
U = u-axis value for the P data for a given voxel location and projection angle
V = v-axis value for the P data for a given voxel location and projection angle

$$U = \frac{T * SID}{(SOD - S)} \qquad V = \frac{z * SID}{(SOD - S)}$$

$$w = \left[\frac{SOD}{(SOD - S)}\right]^2$$

$$S = x\cos\beta + y\sin\beta \qquad T = -x\sin\beta + y\cos\beta$$

Pick the Biggest Target First

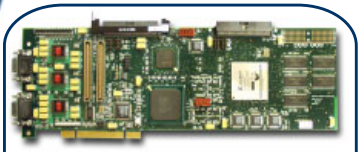
- **Cone Beam Back Projection – 400 G Operations**
  - **Existing optimized PPC code does 250 MFLOPS, compared with the peak 3.2 GFLOPS for a G4 @ 400 MHz.**
    - **This big difference tells us this may be a great target for FPGAs**
    - **Why such a difference?**
      - The dataset is too large to fit in cache
      - The same data must be brought into cache repeatedly to have it in an order that keeps the processor running as efficiently as it can
  - **Can an FPGA do better?**
    - **We undertook an iterative algorithm analysis**
    - **We determined back projection can be intergerized**
      - Greatly improves the ops FPGAs can do
    - **We envisioned a candidate architecture that looked promising**
      - A pipeline that organizes the “inner loop” so that repetitive fetching of data from external memory into the FPGA is minimized
      - Estimated performance on our FPGA compute node
      - Recognized that architecture scales inter FPGA and intra FPGA
  - **Ultimately the FPGA architecture ran at 13 GOPS on a 3M gate FPGA compared with the peak 38 GOPS for 16 bit integers that we might expect from such an FPGA.**
    - **An excellent utilization factor**
    - **We kept the FPGA busy 100% of the time**

Shepp-Logan & Decimation

- **Shepp-Logan Filter – 4.5B Ops**
  - Existing optimized PPC code
    - Runs at 980 MFLOPS compared with the peak 3.2 GFLOPS for a G4 @ 400 MHz
    - 512 point real 1-D fast convolution on 150K lines
    - Not I/O limited
    - Runs well on PPC
  - FPGA implementation
    - More ops to directly do convolution
    - Or FP to do fast convolution
    - Best ops/gate design utilizes a high % of gates but is utilized a low % of the time
  - Less than 2% of total ops
  - Conclusion: run Shepp-Logan filter on PPC
- **Interpolated Decimation – 0.13B Ops**
  - PPC implementation
    - About one op per input voxel
    - Single PPC will be I/O limited, but still faster than other tasks
    - Convenient for output data to go through a PPC
    - Volatile code driving display is easy to change on a PPC
  - FPGA implementation
    - Could be added to output of back project task
    - Creates more fabric traffic because the undecimated volume must also be output
    - Harder to change code
  - Less than 0.1% of total ops
  - Conclusion: run interpolated decimation on PPC

System Improvements

- Significant system performance improvements
- Additional performance/watt gains
- Specific target algorithm gains higher



3M Gate FPGA Node

Dual 400-MHz G4 Nodes

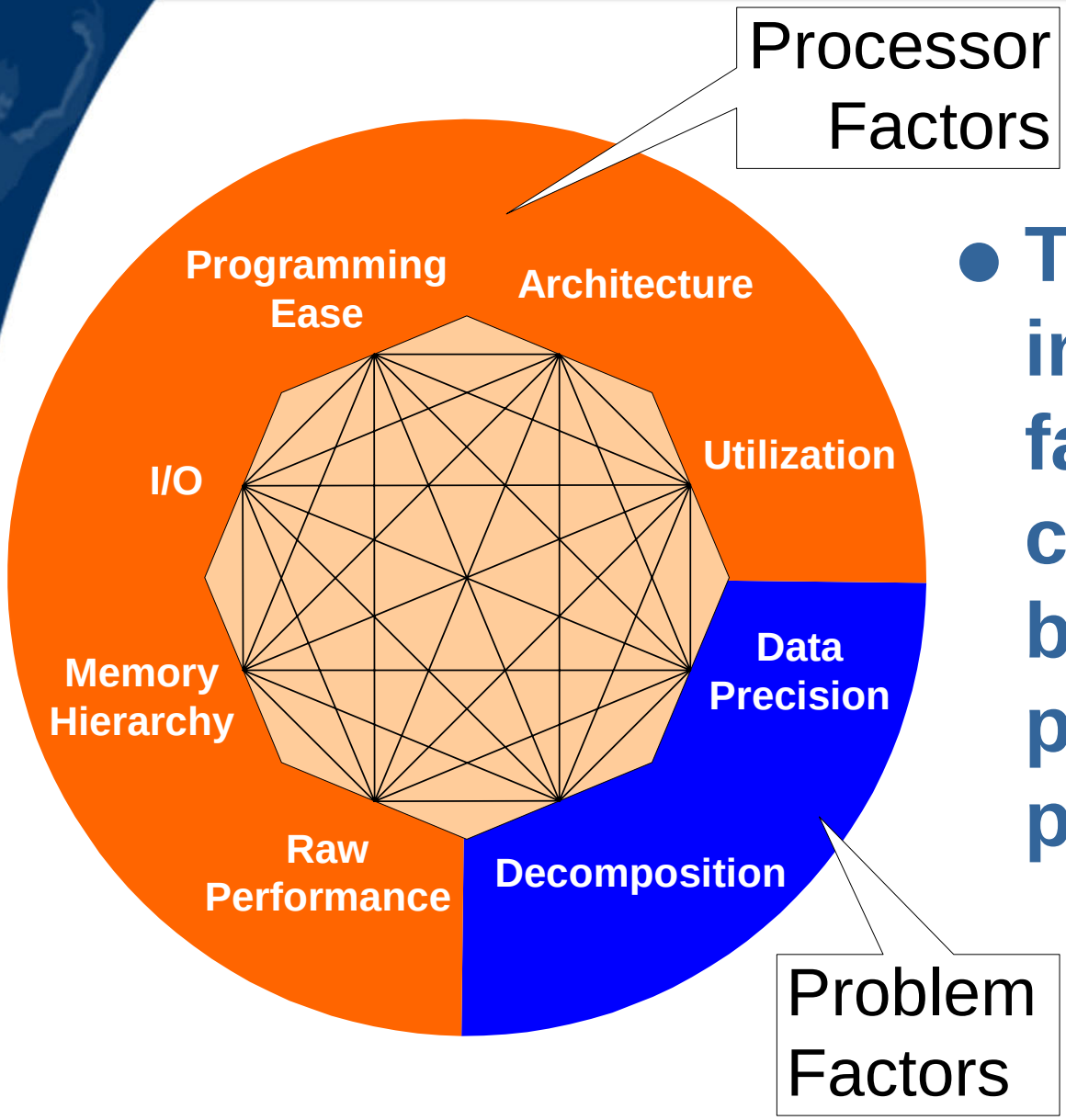
Total System **Back Project Only**

AP-1 Boards	VantageRT 7400 Boards	Render Times	Performance	System Performance per Volume	System Performance per Watt	BackProject Performance per Volume	BackProject Performance per Watt
0	13	60.0▲	1.0	1.0	1.0	1.0	1.0
1	1	28.9◆	2.1	13.5	20.9	26.0	89.4
2	1	15.4◆	3.9	16.9	32.0	24.4	83.9

Normalized to G4 only solution

▲ Estimated value based on optimized AltiVec G4 implementation
◆ Measured value

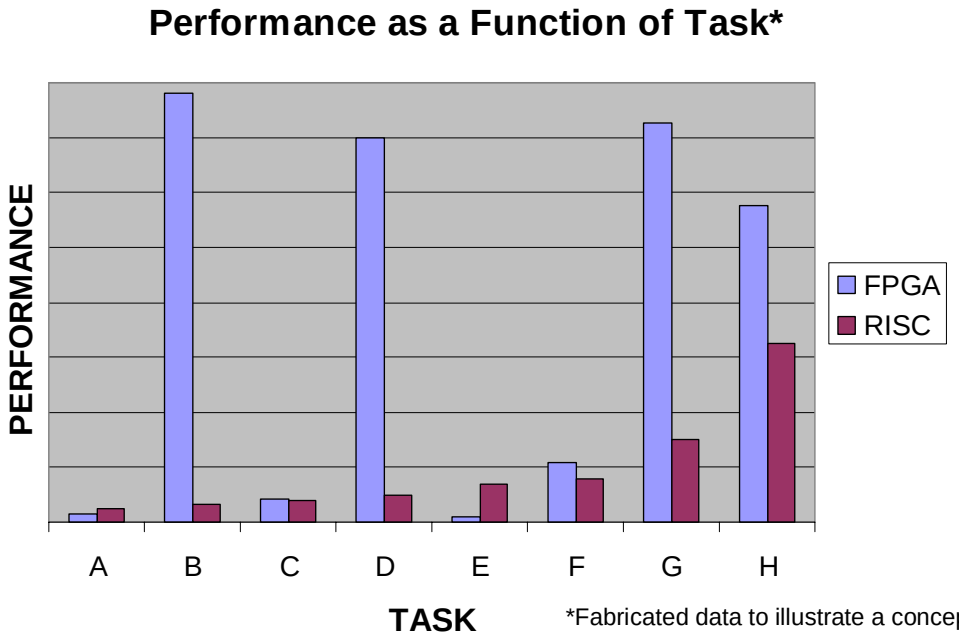
General Factors to Consider



- **These interrelated factors must be considered and balanced when partitioning a problem**

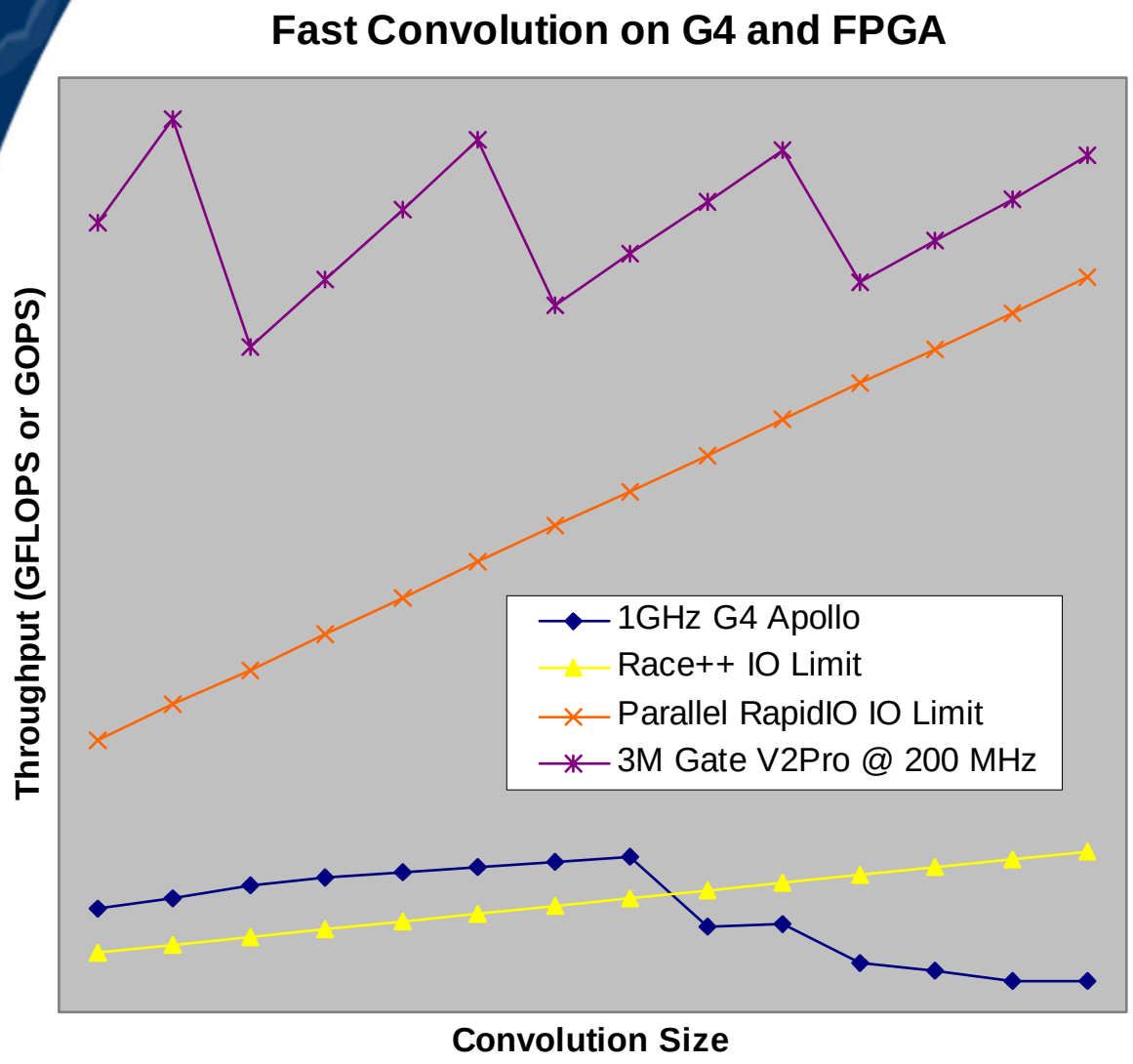
Processor Raw Performance

- Performance may be measured in
  - Operations per second
  - Operations per watt
  - Operations per cubic foot
  - Operations per pound
  - Operations per dollar
- There is not a clear winner
  - Each has different plusses and minuses for each different task



- FPGA raw performance first order rule of thumb
  - 40 to 80% of the hardwired multipliers can be used
    - Interesting FPGAs today have 80 to 400 18 x 18 multipliers that can operate at 100 to 200 MHz
  - There are enough resources around the multipliers to implement the additions, control logic, and so on that the particular task also requires
- PPC with AltiVec raw performance first order rule of thumb
  - For 32-bit floats, the vector unit can retire 4 multiplies and 4 additions per 500 to 1000 MHz clock
  - For 16-bit integers, the vector unit can retire 8 multiplies and 8 additions per 500 to 1000 MHz clock
  - Depending on the vectorizability of the code, 10% to 80% of this number may be achievable

Raw Performance Example



Notes:

- FPGA and G4 performance are representative of anticipated future Mercury products.
- These are well informed estimates that do include concurrent I/O.
- G4 performance falls off dramatically above a threshold because the problem no longer fits in on chip memory.
- FPGA performance does not fall off because the external memory system is fast enough to take over when the problem no longer fits in on chip memory.
- This is one particular architecture to implement FFTs in FPGAs.
- G4 implementation is 32-bit FP.
- FPGA implementation is 32- bit FP I/O with a 27-bit integer minimum internal resolution.

Processor Memory Hierarchy

FPGA

- On Die SRAM
  - Interesting FPGAs today have 80-400 2 KB RAMs @ ~0.8GB/sec Total: 160-800 KB @ 64-320 GB/sec
- External SRAM & DRAM
  - Many choices supported by FPGA supplier, fixed by node design
  - Fixed number of pins available limits total external memory bandwidth
    - Max BW: ~12GB/sec
    - SRAM: ~6 banks @ 2MB each
    - Or DRAM: ~6 banks @ 32MB each
  - Max Storage: constrained only by real-estate and choice of memory
    - BW will suffer as a result of a larger external memory system

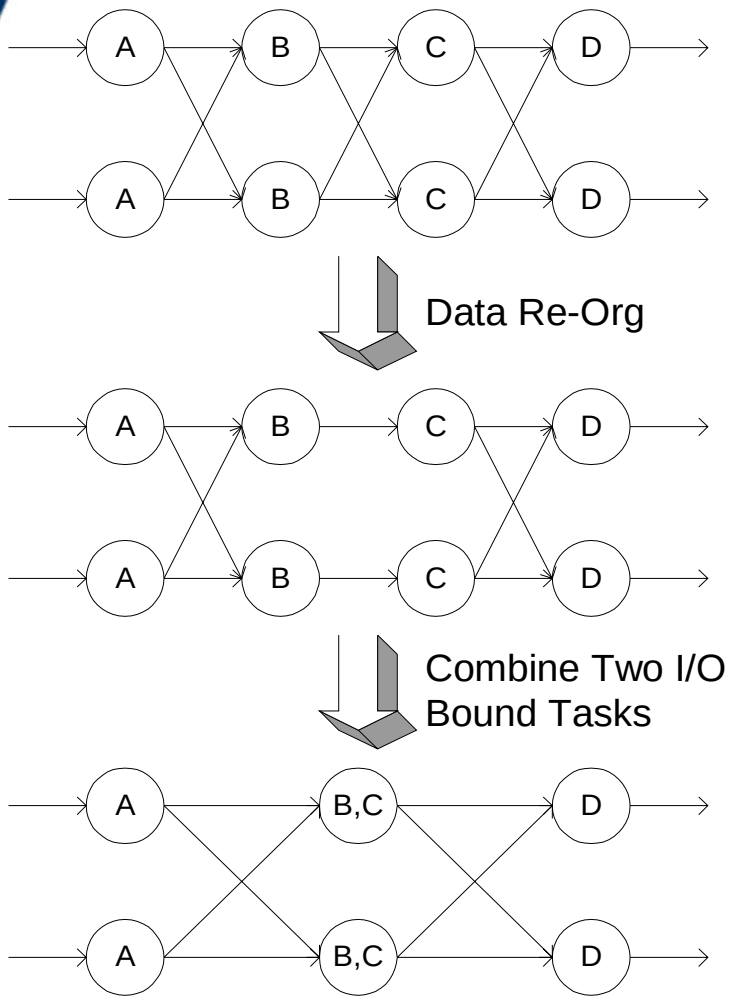
PowerPC

- On Die L1
  - 32/32KB I/D @ 23GB/sec
- On Die L2
  - 256-512KB @ 32GB/sec
- External L3
  - Limited choices offered by processor supplier, fixed by node design
  - 0 1 or 2 MB @ 4GB/sec
- SDRAM
  - Max bandwidth set by processor supplier, constrained by “compute node ASIC,” fixed by node design
  - 0.25-4GB @ 1GB/sec

- Memory hierarchy will affect the performance of most tasks
  - PPC has a fixed general-purpose hierarchy that works well for many tasks
  - FPGA has a malleable memory hierarchy that may be formed to suit the task
    - Much of the FPGA memory hierarchy is not fixed until the application code is written
  - Total memory on a processor node affects how a problem can be decomposed

Processor I/O

Reorganizing to do more work in I/O-bound processors



- I/O per operation rather than raw number of operations can constrain performance.
- RISC processors generally perform I/O functions through their bus or fabric connection.
  - Fixed by processor supplier, or node design
  - 0.26, 1.2, & 2 GB/sec are points for PPCs
- FPGA processors have better raw I/O capability.
  - Fixed by node design
  - Multiple fabric connections to one FPGA are possible
  - I/O external to system may connect directly to FPGA
  - 4 – 8 GB/sec I/O available on today's FPGAs
- When converting a task to FPGA execution provides no benefit because it becomes I/O bound, try changing the decomposition of the problem to combine multiple tasks in one FPGA, thereby reducing the I/O per op.

Processor Programming Ease

- RISC processor tool set is mature
  - Ease of code development, debug, and modification is excellent (relatively speaking)
- Reconfigurable FPGA processor tool set is evolving
  - FPGA architectures themselves do not have the stability and depth of academic understanding that RISC architectures do
  - FPGA programming has many more degrees of freedom than RISC programming
  - For these reasons, development for FPGAs takes many times the effort to develop for RISCs
- This state of affairs leads us to conclude
  - If FPGA and RISC performance for a task are a similar order of magnitude, then choose RISC
  - If the task is a small portion of the overall computing problem such that reducing it will not make a significant system improvement, then choose RISC
  - If task is volatile, in other words the algorithm is not well defined or in flux, then choose RISC

Processor Utilization

FPGA

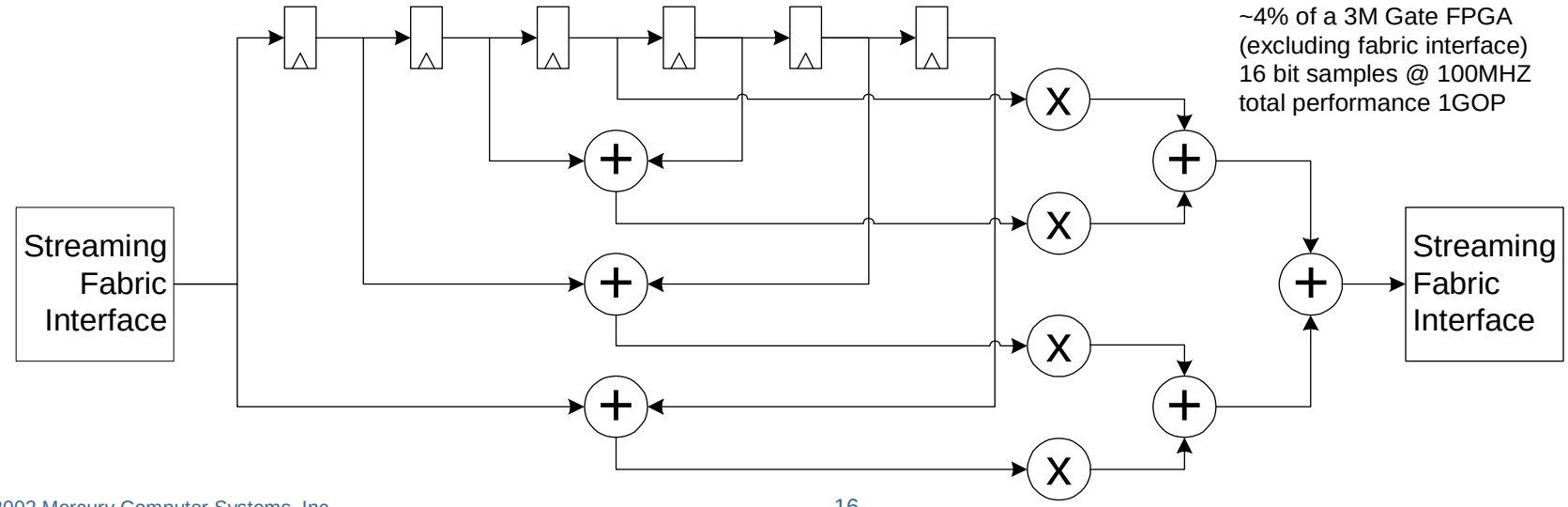
- To maximize FPGA utilization
  - Must program a large portion of the "gates" on an FPGA
  - Must keep all those programmed "gates" working a large portion of the time
- FPGA can be reprogrammed or "context switched" but
  - The whole FPGA goes off-line and looses state during reconfiguration which takes 10s of ms
    - FPGAs some time in the future may work to address these limitations
  - If you context switch often utilization goes down
    - Try to hide FPGA reconfiguration time in human reaction time
- By adding generalizations to FPGA application code utilization can be improved
  - E.G. Write a convolution with settable coefficients instead of fixed ones
  - But, generalization costs gates

RISC

- To maximize RISC utilization
  - Just have to keep it fed with data like any other processor
- RISC can context switch well
  - In 10s of uSec and keep its state
  - Plenty of instructions for different tasks can be kept in memory

Processor Architecture

- RISC processor architecture is fixed by processor supplier
  - We all become experts at this architecture
  - We understand it quantitatively and know how it will respond to changes
- On an FPGA a new processing architecture may be invented for each task
  - Architecture may be designed to optimize performance on a task
  - Architecture may be set by choice of an “FPGA middleware” or a high-level FPGA programming tool
  - My favorite architecture is the streaming dataflow with samples being clocked at the natural clock rate of the FPGA
    - For example, this 7-point symmetrical convolution:



Computer Systems, Inc.
MERCURY
The Ultimate Performance Machine

Problem Decomposition

- **Big problems require multiple processors**
  - There are many different ways to break up problems into tasks for distribution across a homogeneous or heterogeneous set of processors
  - A good partition will optimize system performance
- **Consider radar data, it can be organized by channel, range, or pulse**
  - It may need to be re-organized a number of times in a problem
  - But how can we minimize the re-orging or corner turning?
  - Without enough processing between re-orgs on the fabric, an FPGA may be I/O bound and leave performance on the table
- **Does the problem have a data funnel on the front end?**
  - Is the problem amenable to sensor data being sent directly to FPGA nodes, and processed to the point of a data reduction?
  - If this can be achieved it keeps the voluminous sensor data off the fabric
  - Even without a reduction, there is one less transit of the data across the fabric
- **Is there a choke point in the problem when implemented entirely with RISC processors?**
  - A single task that is inefficient on RISC, or is just a huge number of ops
  - Can a strategic deployment of FPGA nodes on such a task fix the dataflow, and/or reduce the node count?

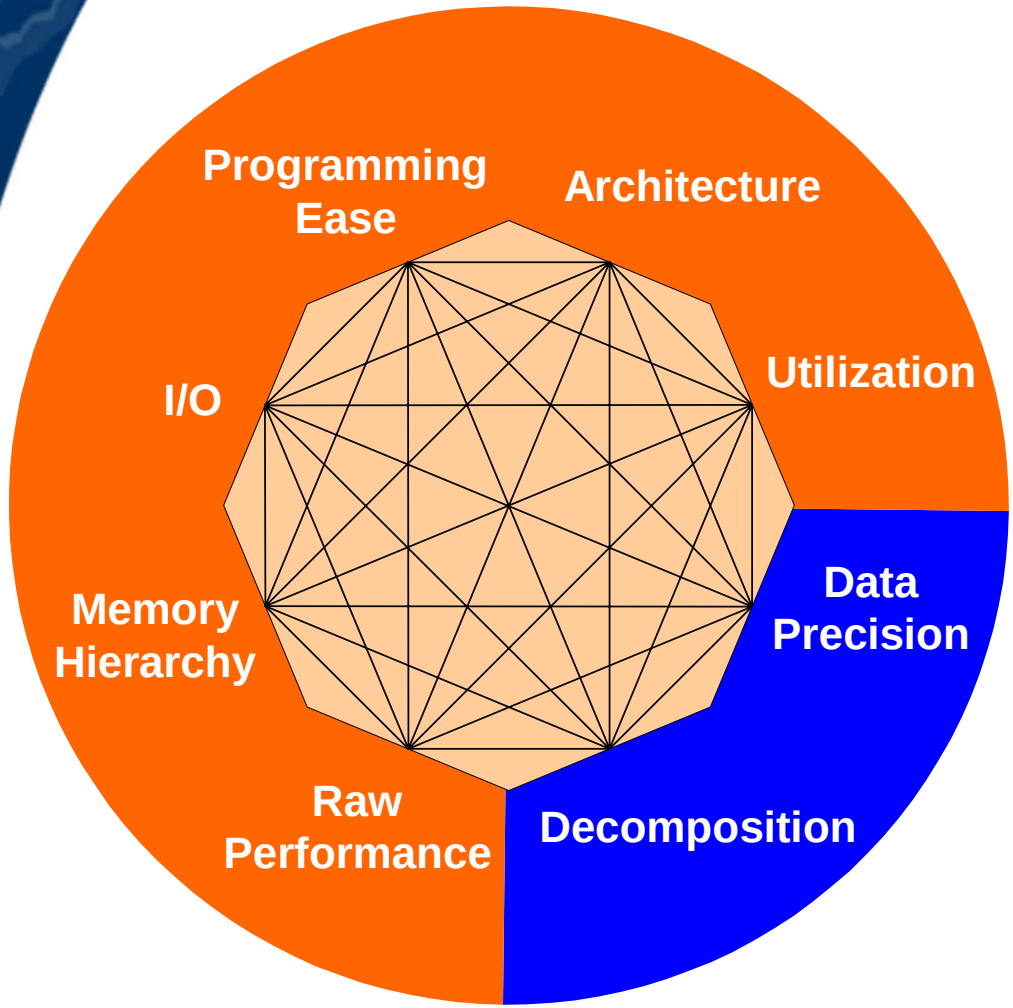
Problem Data Precision

- RISC Processors have a set of supported data types
  - PowerPC with AltiVec can vectorize 32-bit float, and 32-, 16-, and 8-bit ints
    - Because AltiVec performance is the same for 32-bit floats and ints, we use floats for everything
  - Using floats for everything has become the de facto standard because it makes development easier by minimizing the worries about precision
- FPGAs can implement any data type
  - 18 x 18 bit hardwired multipliers and RAMs in 9-, 18-, and 36-bit widths do favor some data types
  - Integer implementations can and should scale the bit precision as needed by each particular portion of an algorithm
    - Minimizing the bit precision of integer processing will maximize the number of ops and operating frequency that can be achieved
  - Floating-point implementations are possible
    - They use more resources per op
    - Many FP resources are coming on line, including an IEEE standards effort
    - FP on FPGAs is not bound to 32- and 64-bit types, any size of FP is possible
  - Bottom line for data precision on FPGAs
    - Engineering the data precision will maximize performance, but costs development effort to manage that precision
    - Some portion of a task that requires FP does not require a trip to the fabric
    - Brute force FP on FPGAs can be done, but performance will be reduced

Future Trends

- Each aspect of the heterogeneous multicomputer is scaling at different rates
  - What you know today as a balanced system will not be in the future
  - RISC processors performance scales bumpily along tracking Moore's law
  - FPGA gate capacity scales with Moore's law, but the operating frequency is also doubling every three years
- FPGAs will get chunkier
  - Memory chunks have been in FPGAs for a while
  - Hardwired multiplier chunks are a recent FPGA feature
  - PPC 405 RISC processor chunks are here now
    - This leads to another set of issues
  - You can expect more DSP chunks

Final Thoughts



- All factors are interrelated
  - Push on one, and often another one pops out
- Identify the largest computational tasks to attack first
- This is the same systems engineering problem you already know how to solve