Mitrion-C Application Development on SGI Altix 350/RC100

Volodymyr V. Kindratenko[1]

Robert J. Brunner[12], Adam D. Myers[2]

1) Innovative Systems Laboratory (ISL)

National Center for Supercomputing Applications (NCSA)

2) Department of Astronomy

University of Illinois at Urbana-Champaign

Research Objectives

- **Evaluate suitability of SGI RASC RC100 platform for scientific computing applications**
  - Focus on applications that require double-precision floating-point arithmetic
- **Evaluate suitability of Mitrion SDK as a development platform for SGI RASC RC100 hardware**
  - Language flexibility
  - Completeness of hardware support
  - Suitability for complex scientific codes

Research Approach

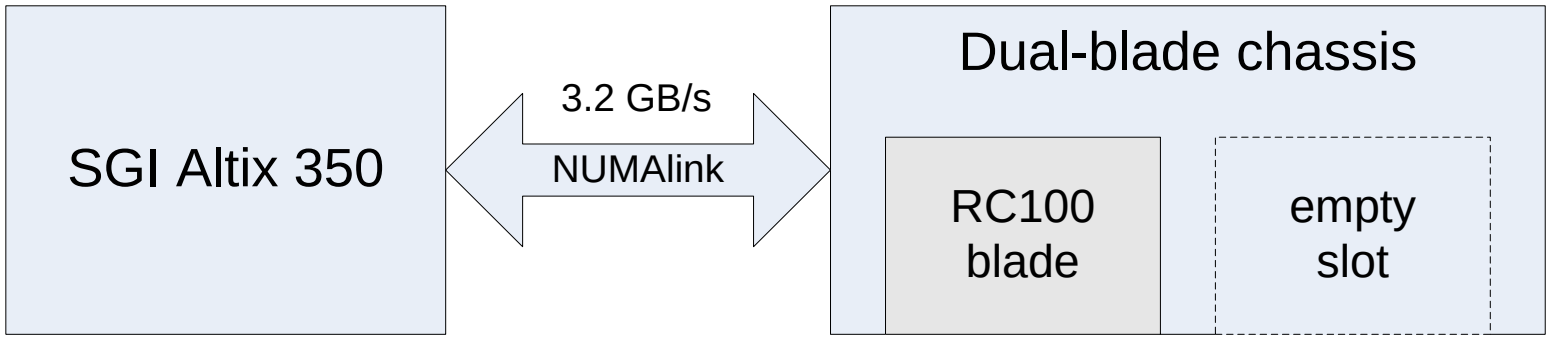
- **Use an existing HPC application as a test case**
  - Cosmology application as the driver
  - With known performance characteristics on other platforms
- **Implement the application in Mitrion-C targeting RC100 platform**
- **Evaluate the implementation, implementation effort, and resulting performance**
  - In relation to the hardware and software used to implement and run the code

Presentation Outline

- **Background information**
  - SGI RASC RC100 platform overview
  - Mitrion-C application development framework
  - Testcase application
- **FPGA kernel design and implementation**
  - Single-core kernel
  - Dual-core kernel
- **Evaluation discussion**
  - SGI RASC RC100 blade
  - Mitrion SDK for RC100

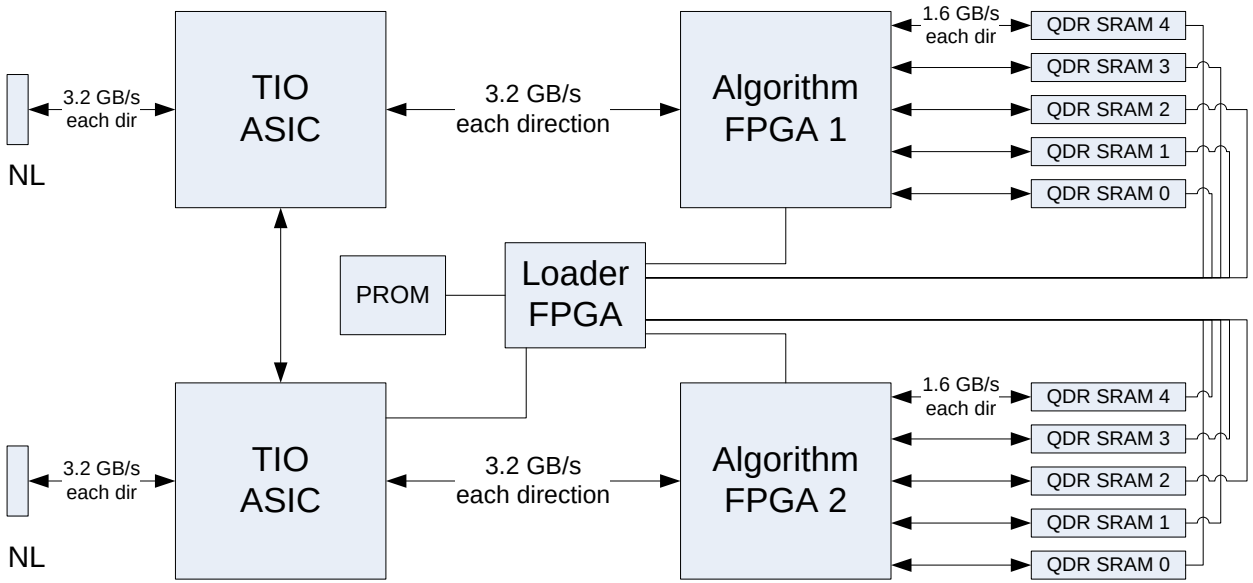
SGI Altix 350 with RC100 blade

- **A standalone, single-module SGI Altix 350**
  - dual-1.4 GHz Intel Itanium 2 system
  - 4 GBs of physical memory.
- **A single dual-blade chassis with one RC100 blade**
  - attached to the host system via a NUMAlink 4 interconnect



SGI RASC RC100 blade

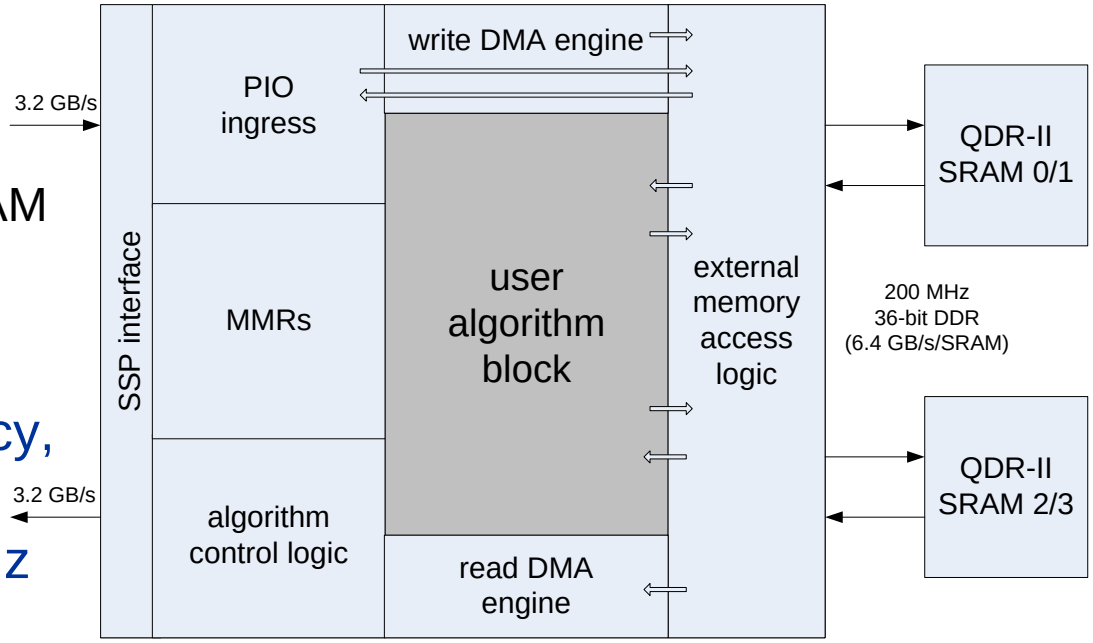
- **Third-generation Reconfigurable Application-Specific Computing (RASC) hardware module**



  - Two computational FPGAs, and a special-purpose loader FPGA
  - Two peer-to-peer I/O (TIO) ASICs
  - Up to 10 QDR SRAM memory modules, configured as two banks to match the NUMAlink 4 channel bandwidth (3.2 Gbyte/sec) to the memories (2x1.6 Gbyte/sec).

RASC RC100 blade FPGA

- **Xilinx Virtex 4 LX200 (XC4VLX200-FF1513-10) chips**
  - 200,448 logic cells
  - 336 Block RAM/FIFOs
    - 6,048 kbits of total BRAM
  - 96 DSP48 slices
  - and 960 user I/O pins
  - maximum clock frequency, as implemented in the RC100 blade, is 200 MHz



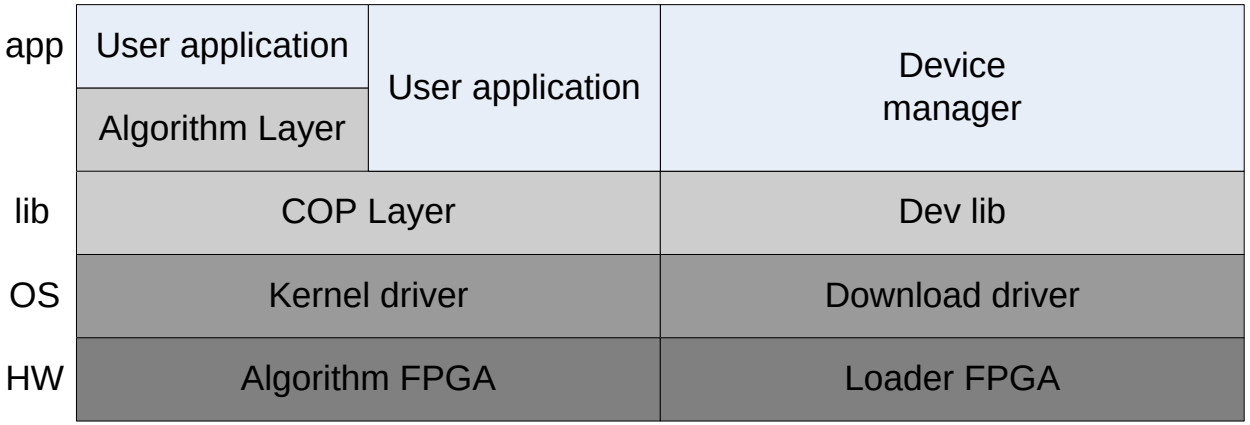
- **A portion of each chip is allocated to the RASC Core Services logic with the rest of the logic allocated to the user algorithm block**

RASC Core Services

- **IP cores provided by SGI**
  - interface between the TIO chip and SRAMs attached to the FPGA
    - user algorithm block has access to two dual-port memory resources, each is 128-bit wide and up to 1 million words deep (16 MB per port per bank)
  - provide memory-mapped register (MMR) inputs and algorithm control logic
    - user algorithm block has direct access to MMR inputs and access to SRAM via external memory logic implemented as part of the RASC Core Services

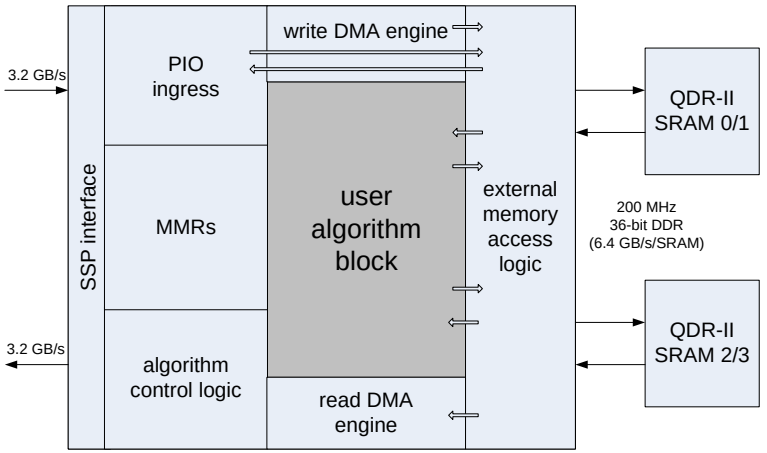
RASC software stack

- **User application interacts with the device either via the co-processor or algorithm layer, their use is mutually exclusive**
  - the Co-Processor (COP) layer: an interface to work with individual devices
  - the algorithm layer, built on top of the COP layer, treats a collection of devices as a single logical device
    - RASC kernel driver
- **Device Manager is a user-space utility for bitstream management**
  - Implemented on top of device library
    - FPGA bitstream download driver



COP layer API usage example

- **// allocate system memory**
  - *char* a0_in[bram_size];
  - *char* b0_in[bram_size];
  - *char* b0_out[bram_size];



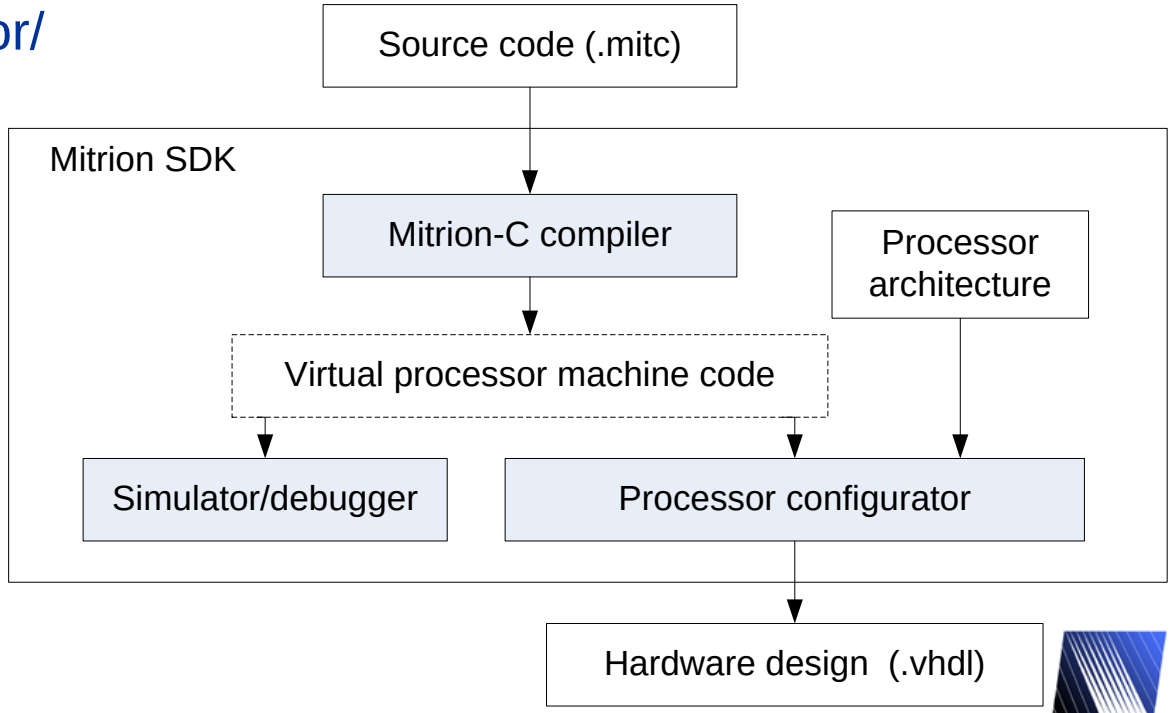
- **// populate with data**
  - *double* *xi = (double*)(a0_in+i*16);
  - *xi = data1[i].x;

- **// do FPGA work**
  - rasclib_cop_send(algorithm_id, "a0_in", a0_in, bram_size);
  - rasclib_cop_send(algorithm_id, "b0_in", b0_in, bram_size);
  - rasclib_cop_go(algorithm_id);
  - rasclib_cop_receive(algorithm_id, "b0_out", b0_out, bram_size);
  - rasclib_cop_commit(algorithm_id, NULL);
  - rasclib_cop_wait(algorithm_id);

- **// process results**
  - *long long* *bin_val = (long long*)b0_out;
  - res = bin_val[2*k+1];

Mitrion SDK for RC100

- **Framework in which applications for RC100 can be implemented**
  - Mitrion-C language
  - Mitrion-C compiler
  - Functional simulator/ debugger
  - Mitrion Virtual Processor and processor configurator
  - Mitrion Host Abstraction Layer (MITHAL) library



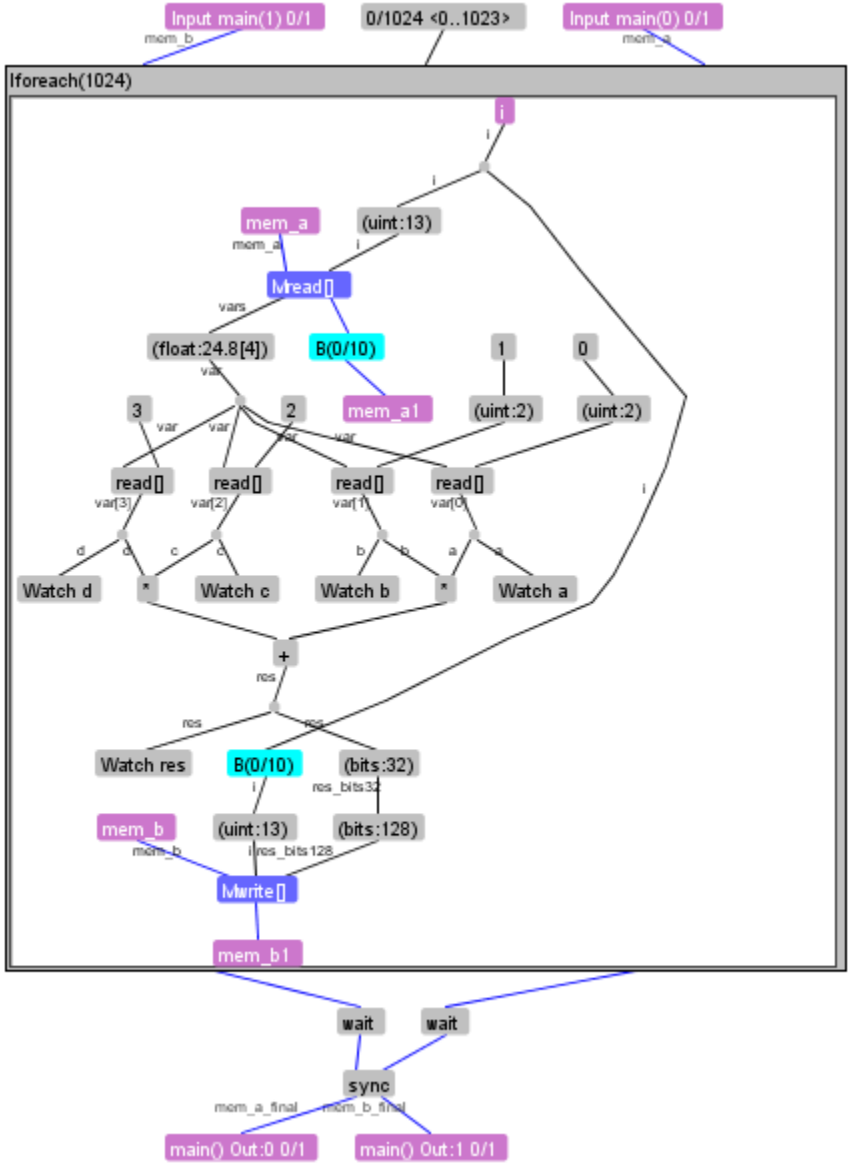
Mitrion-C and Virtual Processor

- **Mitrion-C**
  - Similar to C about as much as Java is similar to C
  - Implicitly parallel
  - Describes data dependencies rather than order-of-execution
  - Allows fine-grain parallelism to be described
  - Syntactic support to make design decisions regarding parallelism
  - Designed to preserve and reveal parallelism in the algorithm
- **Mitrion Virtual Processor**
  - A massively parallel high-performance processor for FPGAs that executes software written in the Mitrion-C programming language
  - Processor's architecture follows a cluster model, placing all processing nodes within an FPGA and creating an ad-hoc network between them

- **Mitrion-C 2D dot product code example**

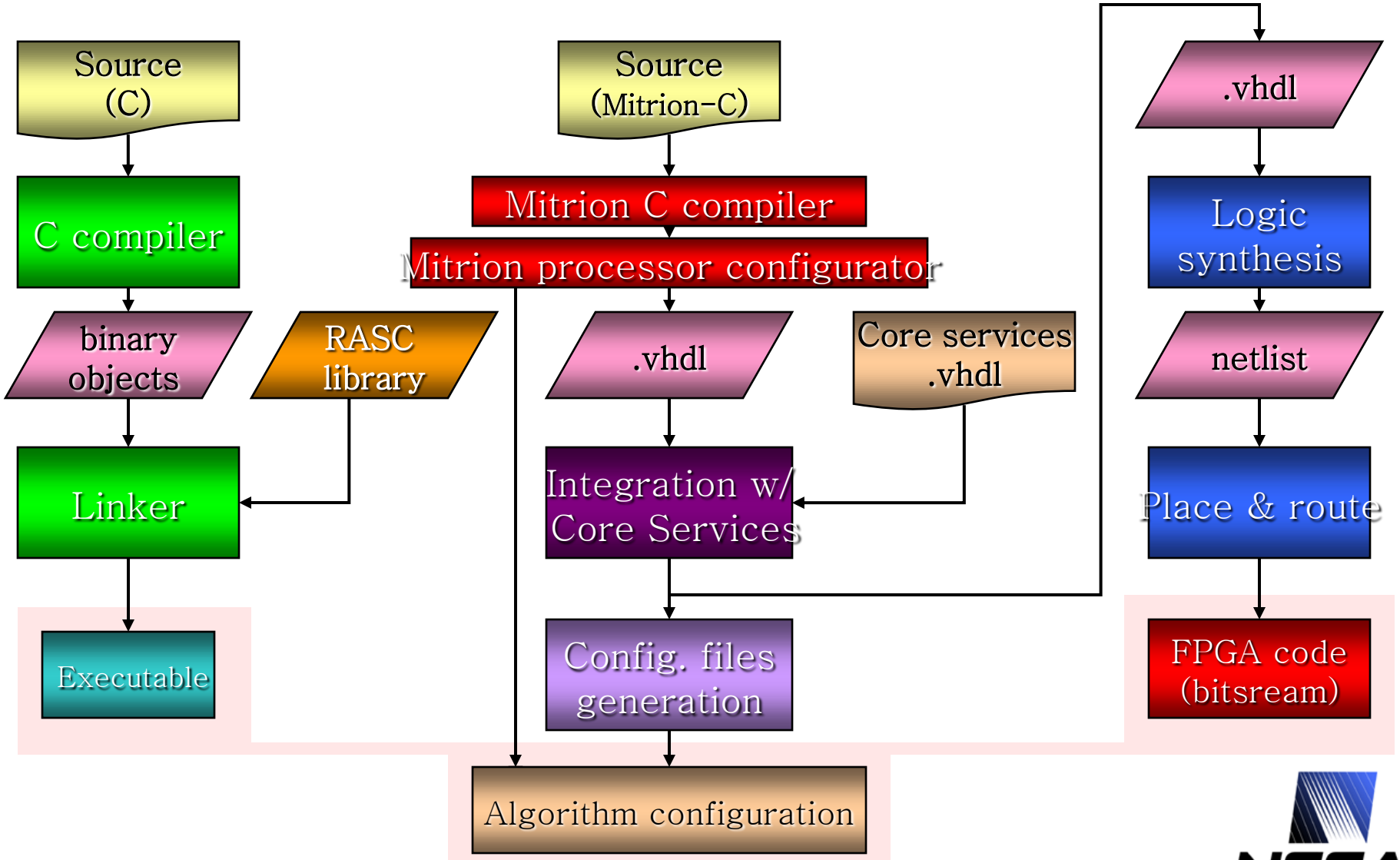
```
Mitrion-C 1.0;
// Options: -cpp
#define NUM_WORDS 1024
#define NUM_WORDS_1 1023
#define ExtRam mem bits:128[8192]

(ExtRam, ExtRam) main(ExtRam mem_a, ExtRam mem_b)
{
   (mem_a1, mem_b1) = foreach(i in <0.. NUM_WORDS_1>)
   {
      (vars, mem_a1) = _memread(mem_a, i);
      float:24.8[4] var = vars;
      a = var[0];
      b = var[1];
      c = var[2];
      d = var[3];
      res = a * b + c * d;
      bits:32 res_bits32 = res;
      bits:128 res_bits128 = res_bits32;
      mem_b1 = _memwrite(mem_b, i, res_bits128);
   } (mem_a1, mem_b1);
   (mem_a_final, mem_b_final) = _wait(mem_a1, mem_b1);
} (mem_a_final, mem_b_final);
```

Mitrion-C Simulator/debugger



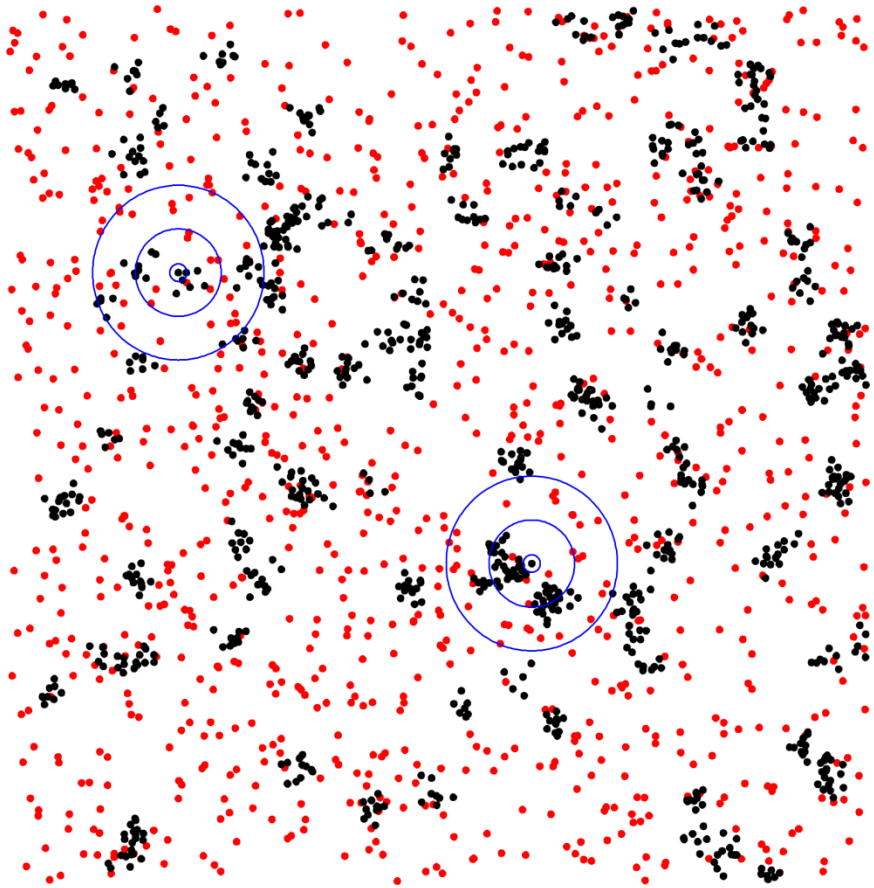

```
---------- Summary: Bandwidth Graph ---------------
ROOT
|  l: 1067.0 ch: 1024 d: 1.042
|    L  limited. Relaxed ch 1067 l 1067
|    body l: 1067 ch: 1024 relaxed ch: 1067
   |--1
   | iterations: 1024   bottleneck: 1.042
   |   main/foreach<1024>
   |  l: 1067.0 ch: 1024 d: 1.042
   |    BW limited. Relaxed ch 1067 l 1067
   |    body ch: 1.0 relaxed ch: 1
-------- End Summary: Bandwidth Graph --------------

----[   Execution time: 10.670us, 1067 steps@100MHz
----[   Executions per second: 93720.712

---------- Summary --------------------------------
Memories:
  External: 8192x128 1 readers and 0 writers.
  External: 8192x128 0 readers and 1 writers.
---------- End Summary ----------------------------

---------- Summary: Hardware resources used --------

  Target FPGA is Xilinx Virtex-II XC2V6000-6FF1517C
  Target Clock Frequency is 100MHz
  18861 Flip Flops out of 67584 = 27% FlipFlop usage
    128 16-bit shiftregisters
     25 BlockRAMs out of 144 = 17% BlockRAM usage
      8 MULT18X18s out of 144 = 5% MULT18X18 usage

-------- End Summary: Hardware resources used -------
```

Mitrion-C/RASC Code Design Flow



Testcase application: TPACF

- **TPACF, denoted as $\omega(\theta)$, is the frequency distribution of angular separations θ between celestial objects in the interval $(\theta, \theta + \delta\theta)$**
  - θ is the angular distance between two points
- **Example**
  - Red Points are, on average, randomly distributed, black points are clustered
  - Red points: $\omega(\theta)=0$
  - Black points: $\omega(\theta)>0$
  - Can vary as a function of angular distance, θ (blue circles)
  - Red: $\omega(\theta)=0$ on all scales
  - Black: $\omega(\theta)$ is larger on smaller scales



TPACF Computational Kernel

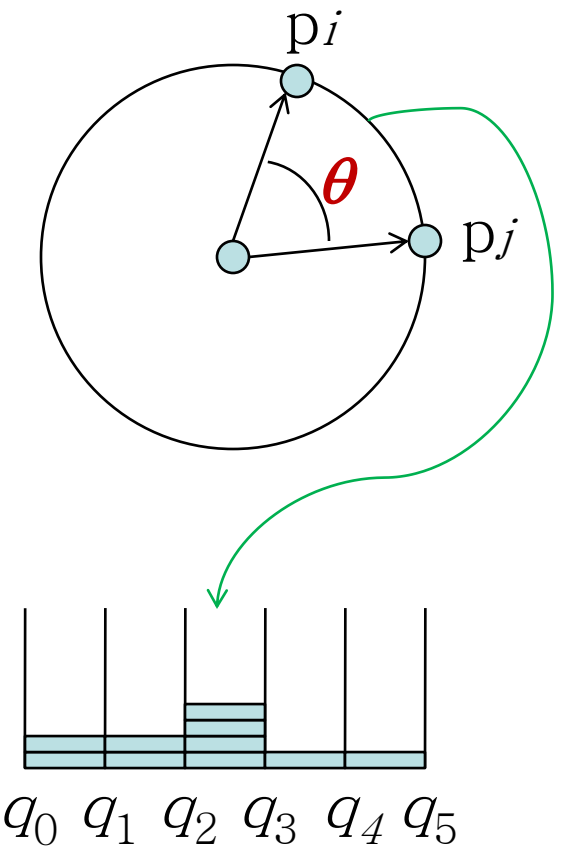
- **The angular correlation function is calculated using the estimator derived by Landy & Szalay (1993):**

$$\omega(\theta) = \frac{DD(\theta) - 2DR(\theta) + RR(\theta)}{RR(\theta)}$$

- **where *DD*(θ) and *RR*(θ) are the angular separation distributions (autocorrelation) of the data and random points, respectively, and *DR*(θ) is the angular separation distribution (cross-correlation) between the data and random points**
- **The problem of computing angular separation distributions can be expressed as follows:**
  - **Input:** Set of points x_1, .., x_n distributed on the surface of a sphere, and a small number *M* of bins: $[q_0, q_1)$, $[q_1, q_2)$, .., $[q_{M-1}, q_M]$.
  - **Output:** For each bin, the number of unique pairs of points (x_i, x_j) for which the angular distance is in the respective bin: $B_l = |\{ij: q_{l-1} <= x_i \cdot x_j < q_l\}|$.

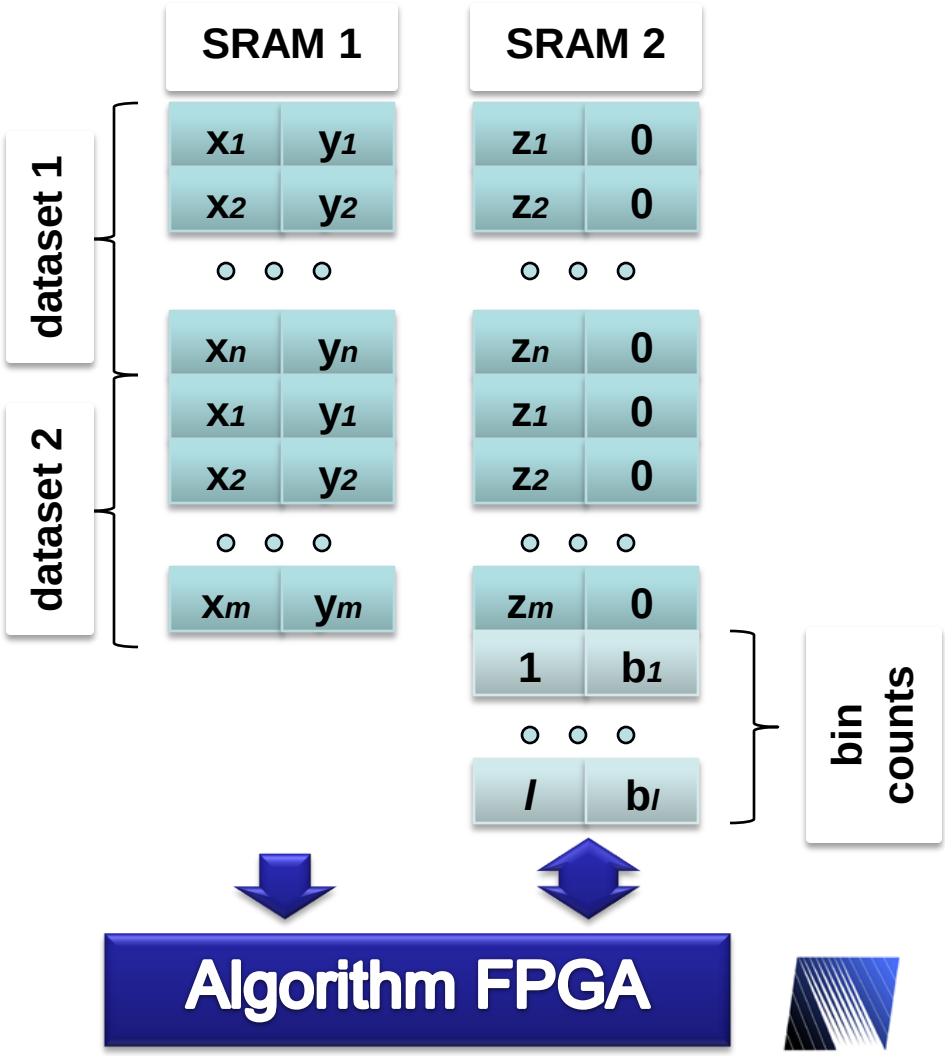
Reference C implementation

```
for (i = 0; i < ((autoCorrelation) ? n1-1 : n1); i++)
{
   double xi = data1[i].x;
   double yi = data1[i].y;
   double zi = data1[i].z;

   for (j = ((autoCorrelation) ? i+1 : 0); j < n2; j++)
   {
      double dot = xi * data2[j].x + yi * data2[j].y + * data2[j].z;

      // binary search
      min = 0;  max = nbins;
      while (max > min+1)
      {
         k = (min + max) / 2;
         if (dot >= binb[k])  max = k;
         else min = k;
      };

      if (dot >= binb[min])  data_bins[min] += 1;
      else if (dot < binb[max]) data_bins[max+1] += 1;
      else data_bins[max] += 1;
   }
}
```

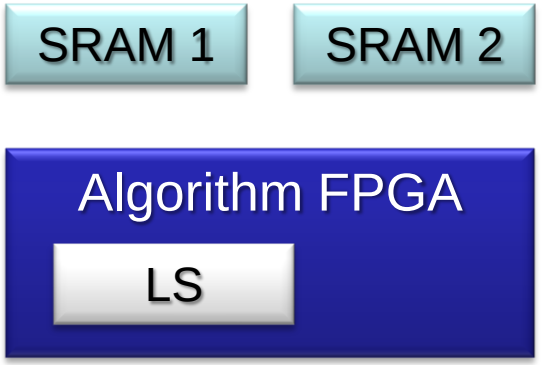


FPGA Kernel Design

- **Data storage/access requirements**
  - Need to be able to read all three coordinates for a given point in one clock cycle
  - Need to fit both datasets in SRAM
  - Need to store output results in SRAM (at the end of the calculations)



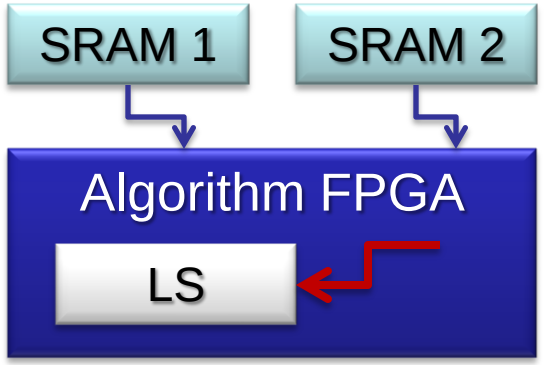
Kernel re-written in Mitrion-C




```
Mitrion-C 1.0;
// options: -cpp
#define ExtRAM mem bits:128[262144]
#define NPOINTS 97160
#define NPOINTS_1 97159
#define NBINS 32

(ExtRAM, ExtRAM) main (ExtRAM a0, ExtRAM b0)
{
   float:53.11[NBINS] binb = [ 0.99999999999576916210, 0.99999999998937272316, 0.99999999997330546453,
         0.99999999993294641509, 0.99999999983156895311, 0.99999999957692020658, 0.99999999893727176126,
         0.99999999733054723006, 0.99999999329463784559, 0.99999998315689186956, 0.99999995769202532081,
         0.99999989372717357217, 0.99999973305473655039, 0.99999932946385972077, 0.99999831568965213968,
         0.99999576920548627346, 0.99998937273599586284, 0.99997330559122843407, 0.99993294712784397404,
         0.99983157364604546835, 0.99957695008220059929, 0.99897345993583771270, 0.99733173469789593302,
         0.99330212814615548300, 0.98320412044889693437, 0.95798951231548890028, 0.89559620527121441835,
         0.74472199710330100331, 0.40112945079596057374, -0.26150795041456115220, -0.97304487057982380627,
        -100.0 ];

   uint:64[NBINS] binsA = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ];

   ExtRAM a3 = a0;
   ExtRAM b3 = b0;
```

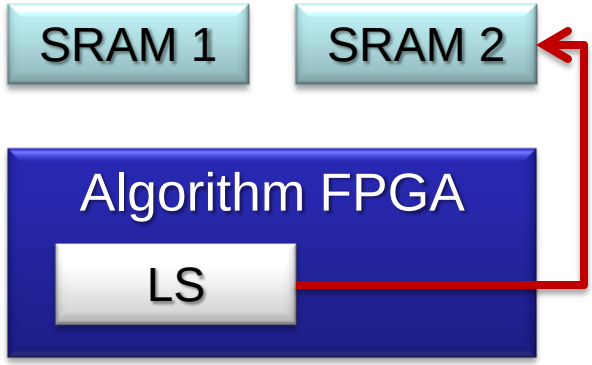
Kernel re-written in Mitrion-C

```
// loop in one data set
(bins, afinal, bfinal) = for (i in <0 .. NPOINTS_1>)
{
   (xi, yi, zi, a1, b1) = readpoint(a0, b0, i); // read next point

   uint:64[NBINS] binsB = binsA;
   ExtRAM a2 = a0;
   ExtRAM b2 = b0;

   (binsA, a3, b3) = for(j in <0 .. NPOINTS_1>)
   {
      (xj, yj, zj, a2, b2) = readpoint(a1, b1, j+NPOINTS); // read next point

      float:53.11 dot = xi * xj + yi * yj + zi * zj; // compute dot product

      int:8 indx = findbin(dot, binb); // find what bin it belongs to

      // update bin
      binsB = foreach (bin in binsB by ind) if (ind == indx) bin + 1 else bin;
   } (binsB, a2, b2);
} (binsA, a3, b3);
```



Kernel re-written in Mitrion-C

```
    ExtRAM bw = b0;
    int:64 idx = 0;
    int:64<NBINS> binsR = reformat(bins, <NBINS>);

    bfinal2 = for (o in binsR)      // write results back to CPU
    {
        bits:128 out_val = [ idx, o ];
        bw = _memwrite(bfinal, idx, out_val);
        idx = idx + 1;
    } bw;
    bdone2 = _wait(bfinal2);
} (afinal, bdone2);
```

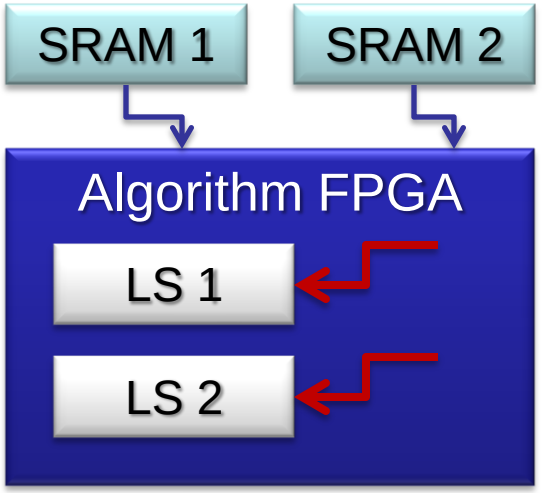


FPGA Resources Utilization

	Total available	Mitrion-C estimate	After place and route
Slices	89,088		42,346 (47%)
Flip Flops	178,176	51,327 (29%)	50,128 (28%)
LUTs	178,176		48,724 (27%)
BlockRAMs	336	29 (8%)	27 (8%)
DSP48s	96	48 (50%)	48 (50%)
P&R time	5 hours 24 minutes 7 seconds		

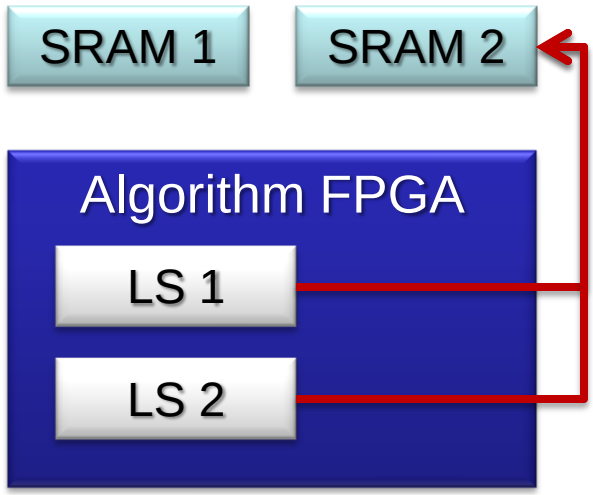
Dual kernel in Mitrion-C

```
(bins1, bins2, afinal, bfinal) = for (i in <0 .. 48579>)
{
   (xi1, yi1, zi1, a0a, b0a) = readpoint(a0, b0, i*2); // read next 2 points
   (xi2, yi2, zi2, a1, b1) = readpoint(a0a, b0a, i*2+1);

   /* ... Omitted code … */

   (binsA1, binsA2, a3, b3) = for(j in <0 .. 97159>)
   {
      // read next point
      (xj, yj, zj, a2, b2) = readpoint(a1, b1, j+97160);

      float:53.11 dot1 = xi1 * xj + yi1 * yj + zi1 * zj;   // compute dot product
      float:53.11 dot2 = xi2 * xj + yi2 * yj + zi2 * zj;

      int:8 indx1 = findbin(dot1, binb); // find what bin it belongs to
      int:8 indx2 = findbin(dot2, binb);

      binsB1 = foreach (bin in binsB1 by ind) if (ind == indx1) bin + 1 else bin; // update bin
      binsB2 = foreach (bin in binsB2 by ind) if (ind == indx2) bin + 1 else bin;

   } (binsB1, binsB2, a2, b2);
} (binsA1, binsA2, a3, b3);
```



Dual kernel in Mitrion-C

```
    // write results back to CPU
    mem bits:128[262144] bw = b0;
    int:64 idx = 0;

    int:64<32> binsR1 = reformat(bins1, <32>);
    int:64<32> binsR2 = reformat(bins2, <32>);

    bfinal2 = for (o1, o2 in binsR1, binsR2)
    {
        int:64 total_count = o1 + o2;
        bits:128 out_val = [ idx, total_count ];
        bw = _memwrite(bfinal, idx, out_val);
        idx = idx + 1;
    } bw;

    bdone2 = _wait(bfinal2);

} (afinal, bdone2);
```



FPGA Resources Utilization

	Total available	Mitrion-C estimate	After place and route
Slices	89,088		70,301 (78%)
Flip Flops	178,176	80,490 (45%)	82,666 (46%)
LUTs	178,176		83,600 (46%)
BlockRAMs	336	29 (8%)	27 (8%)
DSP48s	96	96 (100%)	96 (100%)
P&R time	1 days 11 hours 3 minutes 23 seconds		

Performance

Measured features/ parameters	Reference C implementation
# CPUs	2
# FPGAs	
FPGA exec (s)	
DD time (s)	428.4
DR+RR time (s)	85,859.9
Load/convert (s)	27.4
Total (s)	86,315.7
Overall Speedup	0.6x

Discussion: RC100

- **Xilinx Virtex 4 VLX200 chip**
  - Pros
    - Large FPGA (89,088 logic slices and 6,048 kbits of total Block RAM)
    - Runs at 200 MHz in RC100
  - Cons
    - Have only 96 18x18 hardware multipliers (DSP48 slices)
      - sufficient to implement only 6 double-precision floating-point multipliers using Mitrion-C
      - this is even less than what was available in the FPGAs used in the second-generation RASC boards
    - Of course additional multipliers can be made out of logic cells, however, they require a substantial number of slices
  - Observation: while suitable for applications that require only a few floating-point multiplications, VLX200 chip is not necessary the best choice for scientific computing applications that require double-precision floating-point operations

Discussion: RC100

- **RC100 blade memory architecture**
  - Physical: up to five QDR SRAM DIMMs per FPGA, up to 8 MBs per DIMM
  - Logical: two 128-bit wide banks
  - Pros
    - Balanced NUMAlink channel bandwidth (3.2 Gbyte/sec) to the memories (2x1.6 Gbyte/sec)
  - Cons
    - Practically, only up to 32 MBs per FPGA since only 4 out of 5 memory banks are actually supported by the RASC Core Services
    - Low memory granularity (128-bit words only)
  - Observation: an off-chip memory layout in which there are multiple 64-bit wide memory banks is, in general, more advantageous than the memory layout used in RC100

Discussion: RC100

- **RASC Software Stack**
  - Pros
    - Direct access to hardware via COP layer
    - More abstracted access to hardware via the algorithm layer
      - wide scaling enables multiple FPGAs configured with the same bitstream to work concurrently on the same problem by automatically dividing the data between them
      - an ability to overlap data transfer and calculations via multi-buffering
    - Simple and straightforward API
  - Cons
    - The use of multi-buffering capability requires single-purpose use of each of two logical memory banks
      - Each bank has to be used either for read or write, but not for both
      - Coupled with the fact that there are only two logical memory banks and the low memory granularity, the practical use of this capability is greatly limited
    - No true streaming capability, data always have to go to SRAM before it can be used in the FPGA
      - At least with the version of the tools that we used
      - **It appears that the latest release of RASC Core Services supports this functionality**

Discussion: RC100

- **Miscellaneous**
  - Pros
    - Device manager provides a simple bitstream management capability
    - Extensions to the GNU debugger (gdb) to enable debugging of the process executed in the FPGA(s)
    - Memory-mapped register (MMR) inputs
    - Pre-compiled RASC Core Services IP cores
  - Cons
    - No direct connection between FPGAs on the blade, no shared memory
      - if data is to be passed between two FPGAs, it have to go back to the host and then return to the appropriate SRAM

Discussion: Mitrion SDK

- **Mitrion-C language**
  - Pros
    - High-level programming language, no need to deal with low-level details
    - Yet, some of the low-level details are still available, e.g., arbitrary-sized numerical data types
    - Language designed for parallel programming
      - Pipelining, automatic loop unrolling, data dependency-driven code execution
  - Cons
    - At the time this work was done, the language did not have an efficient support for loops with run-time defined number of iterations
      - **Latest revision of the language includes a new data type, *streams*, that provides the desirable functionality**
    - Some efforts are required to learn how to use the language efficiently

Discussion: Mitrion SDK

- **SDK for RC100**
  - Pros
    - Both IDE and command line-based interfaces
    - Tight integration with RC100 development environment
      - the compiler generates all the necessary configuration files, sets up the complex top-level project and compilation environment, and invokes the downstream compilation process for the RC100 RASC platform – a one button solution from the Mitrion-C source to the FPGA configuration bitstream
    - The latest version of the SDK provides an ability to specify the logic/DSP48 ratio when synthesizing multipliers
    - FPGA resources utilization is estimated during the compile time
    - Code execution time is estimated during the compile time
  - Cons
    - No support for Memory-mapped register (MMR)
    - Runs only at 100 MHz instead of 200 MHz supported by the system
    - FPGA resources utilization estimates sometimes are not accurate
    - Execution time estimates are not always accurate

Conclusions

- **Suitability of SGI RASC RC100 platform for scientific computing applications**
  - Not the best platform for scientific computing applications that involve floating-point multiplications and/or require a substantial SRAM memory bandwidth
- **Suitability of Mitrion SDK as a development platform for SGI RASC RC100 hardware**
  - A workable solution to program the system
  - Incomplete RC100 hardware support
  - Suitable for implementing simple kernels

Acknowledgements

- **Work was funded in part by NSF grant SCI 05-25308 and by NASA grant NNG06GH15G**
- **An SGI Altix 350 system with an RC100 blade was kindly made available by SGI**
  - Special thanks to Matthias Fouquet-Lapar, Tony Galieti, and Dick Riegner, all from SGI, for their help and support with the SGI system
- **Mitrion SDK for RC100 RASC system was kindly provided by Mitrionics AB**
  - Special thanks for Stefan Möhl and Jace Mogill, both from Mitrionics AB, for their help with Mitrion SDK