Accelerating Cosmology Applications

from 80 MFLOPS to 8 GFLOPS in 4 steps

Volodymyr Kindratenko
Innovative Systems Lab (ISL)
National Center for Supercomputing Applications (NCSA)

Robert Brunner and Adam Myers
Department of Astronomy
University of Illinois at Urbana-Champaign (UIUC)
SRC-6 Reconfigurable Computer

SRC Hi-Bar™ 4-port Switch

SNAP™ Memory
μP
PCI-X

dual-Xeon
2.8 GHz, 1 GB memory

MAPC®
Common Memory
MAPE®

Carte™ 2.2

Control FPGA

1.4 GB/s sustained payload

User FPGA 0
Dual-ported Memory
User FPGA 1

2400 MB/s each

National Center for Supercomputing Applications
Cosmology Quest: What is Out There?

Large Scale Structure in the Local Universe

Legend: image shows 2MASS galaxies color coded by redshift (Jarrett 2004); familiar galaxy clusters/superclusters are labeled (numbers in parenthesis represent redshift). Graphic created by T. Jarrett (IPAC/Caltech)

Source: http://spider.ipac.caltech.edu/staff/jarrett/papers/LSS/
Digit{ized|al} Sky Surveys

From Data Drought to Data Flood

1977-1982
First CfA Redshift Survey
spectroscopic observations of 1,100 galaxies

1985-1995
Second CfA Redshift Survey
spectroscopic observations of 18,000 galaxies

2000-2005
Sloan Digital Sky Survey I
spectroscopic observations of 675,000 galaxies

2005-2008
Sloan Digital Sky Survey II
spectroscopic observations of 869,000 galaxies

Sources: http://www.cfa.harvard.edu/~huchra/zcat/
http://zebu.uoregon.edu/~imamura/123/images/
http://www.sdss.org/
Angular Correlation Function

- **TPACF**, denoted as $\omega(\theta)$, is the frequency distribution of angular separations $\theta$ between celestial objects in the interval $(\theta, \theta + \delta\theta)$
  - $\theta$ is the angular distance between two points

- **Blue points** (random data) are, on average, randomly distributed, **red points** (observed data) are clustered
  - Blue points: $\omega(\theta)=0$
  - Red points: $\omega(\theta)>0$

- Can vary as a function of angular distance, $\theta$ (yellow circles)
  - Blue: $\omega(\theta)=0$ on all scales
  - Red: $\omega(\theta)$ is larger on smaller scales

Image source: http://astro.berkeley.edu/~mwhite/
The Method

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

\[
\omega(\theta) = \frac{1}{n_D^2} \cdot DD(\theta) - \frac{2}{n_Dn_R} \sum DR_i(\theta) + 1
\]

\[
\frac{1}{n_R^2} \sum RR_i(\theta)
\]

where \(DD(\theta)\) and \(RR(\theta)\) are the autocorrelation function of the data and random points, respectively, and \(DR(\theta)\) is the cross-correlation between the data and random points.
Auto/cross-correlation Kernel

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

        register int k, min = 0, max = nbins;
        if (dot >= binb[min]) data_bins[min] += 1;
        else if (dot < binb[max]) data_bins[max+1] += 1;
        // run binary search
        else {
            while (max > min+1)
            {
                k = (min + max) / 2;
                if (dot >= binb[k]) max = k;
                else min = k;
            }
            data_bins[max] += 1;
        }
    }
}
Overall Application Organization

// pre-compute bin boundaries, binb

// compute DD
doCompute{CPU|MAP}(data, npd, data, npd, 1, DD, binb, nbins);

// loop through random data files
for (i = 0; i < random_count; i++)
{
    // compute RR
    doCompute{CPU|MAP}(random[i], npr[i], random[i], npr[i], 1, RRS, binb, nbins);

    // compute DR
    doCompute{CPU|MAP}(data, npd, random[i], npr[i], 0, DRS, binb, nbins);
}

// compute w
for (k = 0; k < nbins; k++)
{
    w[k] = (random_count * 2*DD[k] - DRS[k]) / RRS[k] + 1.0;
}
Step 1: Exploit Deep Parallelism

// main compute loop
for (i = 0; i < n1; i++) {
    pi_x = AL[i]; pi_y = BL[i]; pi_z = CL[i]; // point i
    #pragma loop noloop_dep
    for (j = 0; j < n2; j++) {
        dot = pi_x * DL[j] + pi_y * EL[j] + pi_z * FL[j]; // dot product
        // find what bin it belongs to
        select_pri_64bit_32val( (dot < bv31), 31, (dot < bv30), 30,
                                ... (dot < bv02), 2, (dot < bv01), 1, 0, &indx);
        // what bin memory bank to use in this loop iteration
        cg_countceil_32 (1, 0, j == 0, 3, &bank);
        // update the corresponding bin count
        if (bank == 0) bin1a[indx] += 1;
        else if (bank == 1) bin2a[indx] += 1;
        else if (bank == 2) bin3a[indx] += 1;
        else bin4a[indx] += 1;
    }
}
Unrolling Binary Search Loop

• Original C implementation

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

• Pipelined MAP C implementation

```c
select_pri_64bit_32val( (dot < bv31), 31,
    (dot1 < bv30), 30, (dot1 < bv29), 29,
    (dot1 < bv28), 28, (dot1 < bv27), 27,
    (dot1 < bv26), 26, (dot1 < bv25), 25,
    (dot1 < bv24), 24, (dot1 < bv23), 23,
    (dot1 < bv22), 22, (dot1 < bv21), 21,
    (dot1 < bv20), 20, (dot1 < bv19), 19,
    (dot1 < bv18), 18, (dot1 < bv17), 17,
    (dot1 < bv16), 16, (dot1 < bv15), 15,
    (dot1 < bv14), 14, (dot1 < bv13), 13,
    (dot1 < bv12), 12, (dot1 < bv11), 11,
    (dot1 < bv10), 10, (dot1 < bv09), 9,
    (dot1 < bv08), 8,  (dot1 < bv07), 7,
    (dot1 < bv06), 6,  (dot1 < bv05), 5,
    (dot1 < bv04), 4,  (dot1 < bv03), 3,
    (dot1 < bv02), 2,  (dot1 < bv01), 1,
    0, &indx);
```
Updating Bin Counts

• Original C implementation

```c
data_bins[indx] += 1;
```

– Issue
  • loop slowdown by 4 clocks due to BRAM read/write

– Solution
  • use multiple BRAMs and add results at the end

• Pipelined MAP C implementation

```c
#pragma loop nol loop dep
{
  // what memory bank to use in this iteration
  cg_count ceil 32 (1, 0, j == 0, 3, &bank);

  // update the corresponding bin count
  if (bank == 0) bin1a[indx] += 1;
  else if (bank == 1) bin2a[indx] += 1;
  else if (bank == 2) bin3a[indx] += 1;
  else bin4a[indx] += 1;
}

  // add results at the end
  bin[indx] = bin1a[indx] + bin2a[indx] + …
```
Step 2: Exploit Wide Parallelism

User FPGA 0
- BRAM
- CE 1
- BRAM
- CE 2

User FPGA 1
- merge
- BRAM
- CE 3
- BRAM
- CE 4

OBM A
OBM B
OBM C
OBM D
OBM E
OBM F

1 GFLOP
1 GFLOP
2 GFLOPS
2 GFLOPS
Using both FPGAs

• Primary FPGA

for (i = 0; i < n1; i += 4)
{
    // sync, no OBM permissions granted at this time
    send_perms(0);

    // read NofCALCS_C/2 points from the first dataset
    for (k = 0; k < 2; k++)
    {
        offset = i + k;
        p1x = p2x;
        p2x = AL[offset];
        p1y = p2y;
        p2y = BL[offset];
        p1z = p2z;
        p2z = CL[offset];
    }

    // let other chip to do the same
    send_perms(OBM_A | OBM_B | OBM_C);
}

• Secondary FPGA

for (i = 0; i < n1; i += 4)
{
    // sync, no OBM permissions granted at this time
    recv_perms();

    // sync, OBM permissions are granted for A,B,C
    recv_perms();

    // sync, OBM permissions granted for A,B,C
    recv_perms();
Using both FPGAs

• Primary FPGA

• Secondary FPGA

\[
\text{for (i = 0; i < n1; i += 4)}
\]
\[
\{
\text{ // read NofCALCS\_C/2 points from the first dataset}
\text{ for (k = 2; k < 4; k++)}
\}
\]
\[
\text{ offset = i + k;}
\text{ p1x = p2x;}
\text{ p2x = AL[offset];}
\text{ p1y = p2y;}
\text{ p2y = BL[offset];}
\text{ p1z = p2z;}
\text{ p2z = CL[offset];}
\]

// sync, no OBM permissions granted at this time
send_perms(0);

// sync, no OBM permissions granted at this time
recv_perms();
Using both FPGAs

- **Primary FPGA**

  ```
  #pragma src parallel sections
  {
    #pragma src section
    {
      #pragma loop noloop_dep
      for (j = j_start; j < n2; j++)
      {
        put_stream_dbl(&S5, DL[j], 1);
        put_stream_dbl(&S6, EL[j], 1);
        put_stream_dbl(&S7, FL[j], 1);

        dot1 = p1x * DL[j] + p1y * EL[j] + p1z * FL[j];
        select_pri_64bit_32val( (dot1 < bv31), 31, … // update bin…

        dot2 = p2x * DL[j] + p2y * EL[j] + p2z * FL[j];
        select_pri_64bit_32val( (dot2 < bv31), 31, … // update bin…
      }
    }
  }
  ```

- **Secondary FPGA**

  ```
  #pragma src parallel sections
  {
    #pragma src section
    {
      double DL_j, EL_j, FL_j;

      #pragma loop noloop_dep
      for (j = j_start; j < n2; j++)
      {
        get_stream_dbl(&S2, &DL_j);
        get_stream_dbl(&S3, &EL_j);
        get_stream_dbl(&S4, &FL_j);

        dot1 = p1x * DL_j + p1y * EL_j + p1z * FL_j;
        select_pri_64bit_32val( (dot1 < bv31), 31, … // update bin…

        dot2 = p2x * DL_j + p2y * EL_j + p2z * FL_j;
        select_pri_64bit_32val( (dot2 < bv31), 31, … // update bin…
      }
    }
  }
  ```
Using both FPGAs

• Primary FPGA

```c
#pragma src section
{
    stream_bridge(&S5, STREAM_TO_PORT,
                  BRIDGE_A, d_count);
}

#pragma src section
{
    stream_bridge(&S6, STREAM_TO_PORT,
                  BRIDGE_B, d_count);
}

#pragma src section
{
    stream_bridge(&S7, STREAM_TO_PORT,
                  BRIDGE_C, d_count);
}
```

• Secondary FPGA

```c
#pragma src section
{
    stream_bridge(&S2, PORT_TO_STREAM,
                  BRIDGE_A, d_count);
}

#pragma src section
{
    stream_bridge(&S3, PORT_TO_STREAM,
                  BRIDGE_B, d_count);
}

#pragma src section
{
    stream_bridge(&S4, PORT_TO_STREAM,
                  BRIDGE_C, d_count);
}
```
Step 3: Exploit Fixed-point Arithmetic

- Bin Boundaries
  - From 0.01 to 10,000 arcmin
  - Bin boundaries (in dot product space):
    - 0.999999999995769
    - 0.999999999989373
    - 0.999999999973305
    - 0.999999999932946
    - 0.999999999831569
    - 0.999999999576920
    - 0.999999998937272
    - 0.999999997330547
    - 0.999999993294638
    - ...
  - Only 12 digits after the decimal point are important
    - 41-bit fixed-point will do it!

- Converting to int64
  \[
  \text{int64 dot} = (px*DL_j + py*EL_j + pz*FL_j) \times 1e12;
  \]

- Compare only the lower 40 bits

```
module less40 (A, B, Q, CLK);
  input [63:0] A;
  input [63:0] B;
  output Q;
  input CLK;

  reg Q;

  always @ (posedge CLK)
  begin
  end

endmodule
```
Step 3: Exploit Fixed-point Arithmetic
Step 4: Scale up to multiple MAPs

MAP Series C processor

MAP Series E processor

3.5 GFLOPS

5 GFLOPS

National Center for Supercomputing Applications
For more details: “Dynamic load-balancing on multi-FPGA systems: a case study”, RSSI’07
Measured Performance

FLOPS

Execution time

Performance (MFLOPS)

CPU dual-MAP

8.01 GFLOPS

83 MFLOPS

Execution time (s)

dataset size
SDSS DR5 LRS Analysis Example

- SDSS DR5 photometric-selected Luminous Red Galaxy sample
  - Observed dataset consisting of 1,641,323 points
  - 100 random datasets, 1,000,000 points each

- Model
  - Error estimation using 10 subsets

- Compute time
  - 10.2 days (vs. 980 days on a single 2.8 GHz Intel Xeon chip)
Power and Cost Benefits

• Power savings
  – dual-MAP system, including the dual-CPU motherboard and hardware interfaces necessary to drive the MAPs, consumes about 290 Watt
  – a 90-CPU cluster consisting of 45 dual-Xeon nodes that theoretically could deliver the same application performance as our dual-MAP system, consumes about 9,000 Watt
  – the FPGA-based solution uses 3.2% of the power of the CPU-based solution

• System acquisition cost
  – dual-MAP system was acquired in 2006 for about $100K, which would have been comparable to the cost of a 90-CPU cluster consisting of 45 dual-Xeon nodes
  – new generation SRC-7 MAPStation with a single Series MAP H reconfigurable processor has more capability than our dual-MAP system and costs only half as much, but so are modern CPU-based systems…
Four Steps to Performance

The Recipe

• Step 1: exploit deep parallelism
  – Double-precision floating point arithmetic
  – One chip
  – One fully pipelined compute engine
  – 500 MFLOPS (compute-only)

• Step 2: exploit wide parallelism
  – Double-precision floating point arithmetic
  – Two chips
  – Four compute engines
  – 2 GFLOPS (compute-only)

• Step 3: exploit fixed-point numerical types
  – Fixed-point arithmetic
  – Two chips
  – Seven compute engines
  – 3.5 GFLOPS (compute-only)

• Step 4: scale up to multiple MAPs (MAP C + MAP E)
  – Fixed-point arithmetic
  – Four chips
  – Seventeen compute engines
  – Dynamic job scheduling
  – 8.5 GFLOPS (compute-only)
  – 8.01 GFLOPS (measured including data transfer overhead)
Conclusions

• Reconfigurable computer enabled us to run calculations that require either a very long time to run on a single-processor system, or a large HPC system to run in an acceptable amount of time

• Programming can be done in C language (with some knowledge of the underlying hardware)

• Power savings are likely to be very significant

• System acquisition cost is likely to be comparable to the cost of a similar (performance-wise) conventional HPC system
Acknowledgements

• This work is funded by NASA Applied Information Systems Research (AISR) award number NNG06GH15G
  – Prof. Robert Brunner and Dr. Adam Myers from UIUC Department of Astronomy

• NCSA Collaborators
  – Dr. Rob Pennington, Dr. Craig Steffen, David Raila, Michael Showerman, Jeremy Enos, John Larson, David Meixner, Ken Sartain

• SRC Computers, Inc.
  – David Caliga, Dr. Jeff Hammes, Dan Poznanovic, David Pointer, Jon Huppenthal

National Center for Supercomputing Applications
Reconfigurable Systems Summer Institute (RSSI) 2008

- July 7-10, 2008
- National Center for Supercomputing Applications (NCSA), Urbana, Illinois
- Organized by