Application and Hardware Analysis To Predict Performance

Craig P. Steffen

NCSA Innovative Systems Laboratory

Reconfigurable Systems Summer Institute

July 20, 2007

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

NCSA
National Center for Supercomputing Applications

What question is asked?

We've heard about these great things with FPGAs. What can they do for my problem?

What can they do *FOR ME*?

What question is asked?

We’ve heard about these great things with FPGAs. What can they do for my problem?

What can they do *FOR ME*?

How much effort will it cost to get the best performance out of it?

What question is asked?

We've heard about these great things with FPGAs. What can they do for my problem?

What can they do *FOR ME*?

How much effort will it cost to get the best performance out of it?

How much effort will it cost to find out if it's worth porting my algorithm at all?

What’s Different About FPGAs? Which of these things apply to MY algorithm, in good or bad ways?

- Different programming styles
- Slower clock speeds
- Slower clocks→lower incoming bandwidths
- Wide parallelism
- Deep parallelism

The Real Question:

Is it likely enough that FPGA systems will help my code for someone to spend their time to understand the potential wide or deep parallel aspects it with respect to FPGAs?

NO ⟵ ? ⟶ **MAYBE**

Which items will cause a NO answer?

What are the *inherent* problems in a code/hardware system?

Assuming the FPGA co-processor in a system is infinitely capable and programming is not an issue, what are the limitations of processing speed?

Can we *feed* the abstract super-FPGA fast enough? Considerations:

- Problem size
- Memory bandwidths
- Number of memory banks
- Memory speeds
- Latencies
- Algorithm efficiency (less data better)

From the Domain Expert

- Problem size
- Algorithm efficiency (less data better)

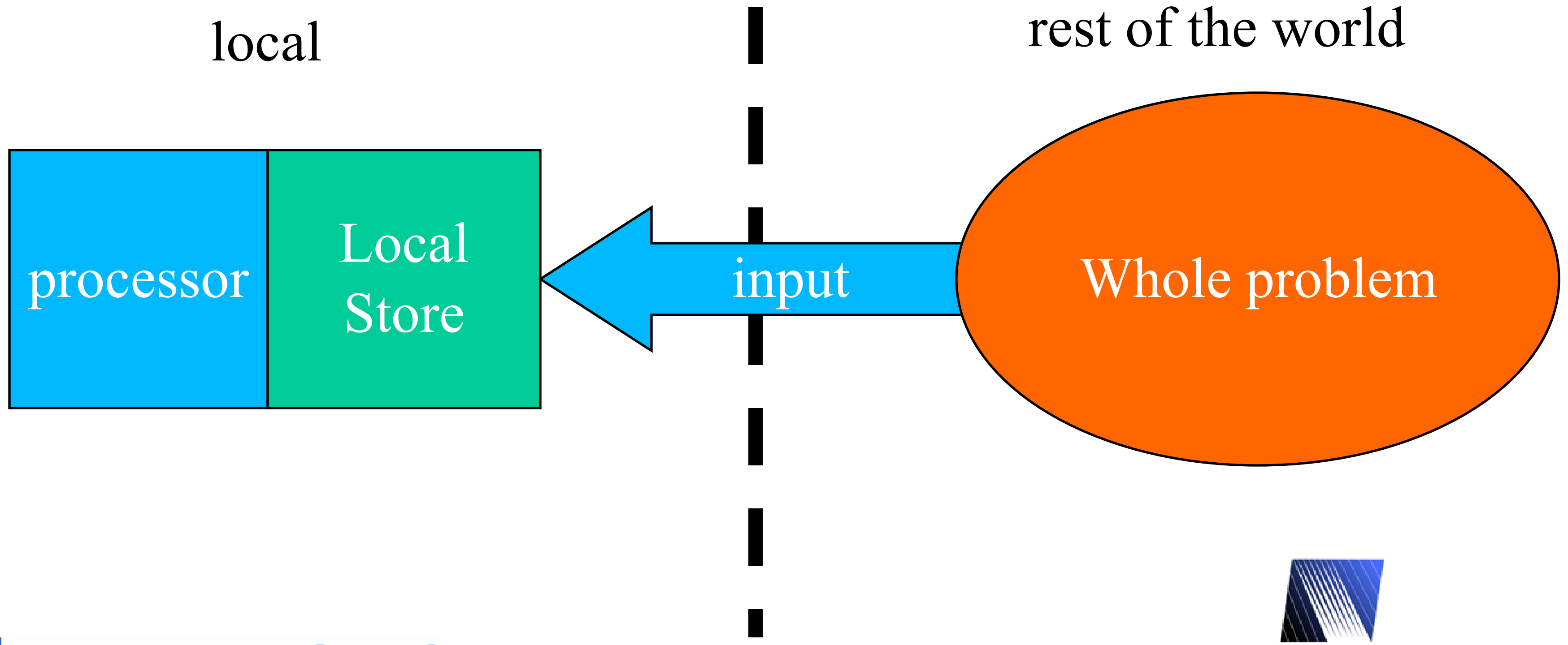
How do we take knowledge of the hardware (which we have) and knowledge of the algorithm (which the expert has) and put them together to answer the NO/MAYBE question? How can this be *quantified*?

Model to Determine the speed of the algorithm:

- Finite memory bandwidths limit the incoming data and so constrain the maximum processing speed
- More than one layer of memory, more than one stage from RAM to processor
- MILC collaboration says that a machine must be able to sustain a certain “bytes per flop”. How does that apply to systems with multiple layers? How do we apply that to arbitrary codes?

Model Assumptions

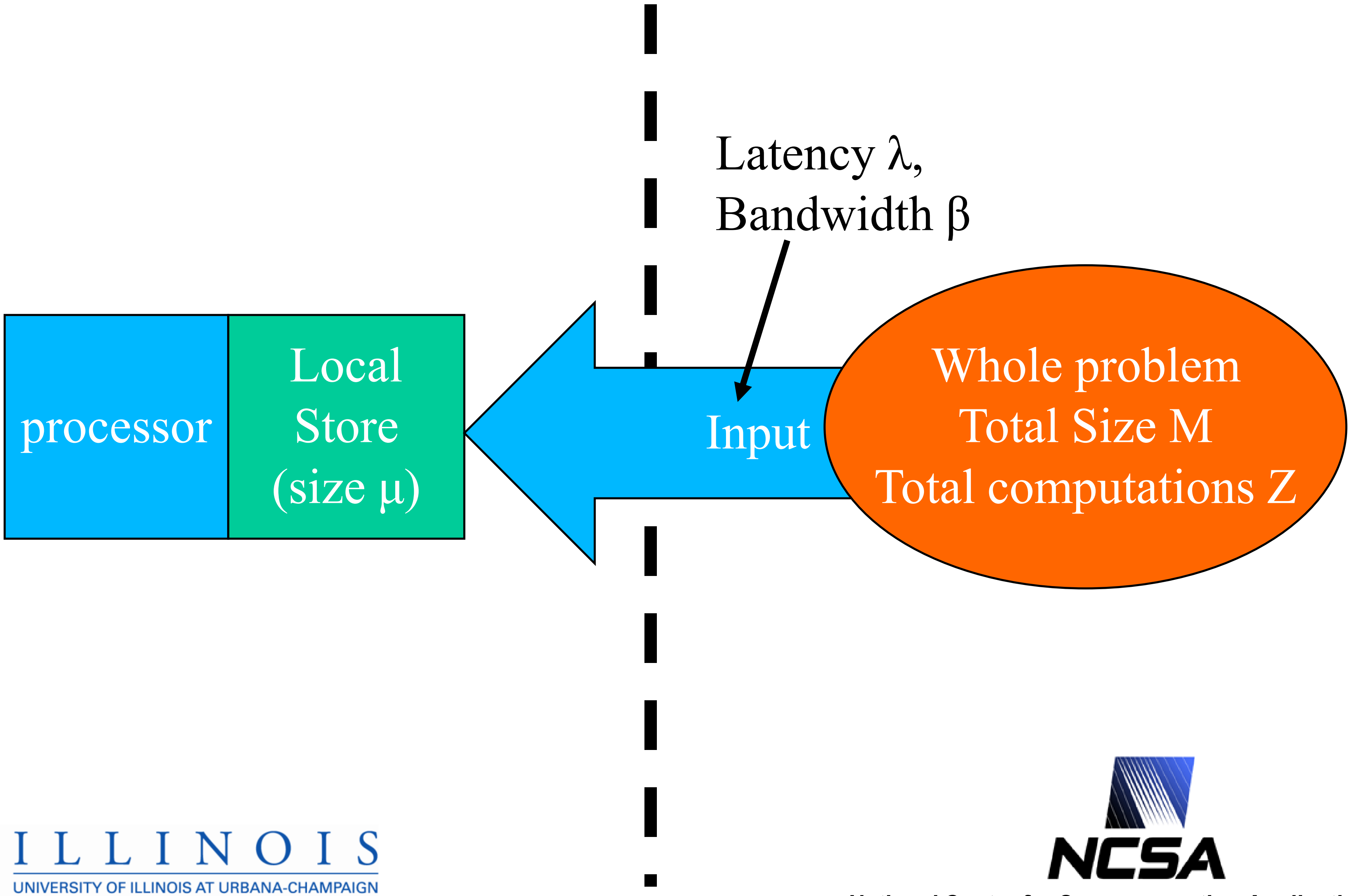
- Arbitrarily fast abstract processor
- Local store is fed through limited bandwidth from outside storage



Model Ignores:

- Temporary storage
- Outputs
- Data packaging time
- Programming difficulty

Model Hardware Parameters



Total Time to Calculate

- Total computational time cannot be less than total load time to load data M
- Total time *may be* greater if we need to load data more than once
- Characterize algorithm/code by computational density; number of computations per byte of input

Computational Density Function

- Computational density is the operations per byte of local store size
- η(α) is the number of computations possible with a local store of size α
- ρ(α) is then the "computational density" the number of computations per byte. This is the efficiency of the computation in terms of cost of moving the data in
- This assumes independence of steps in the calculation (more later)

$$\rho(\alpha) = \frac{\eta(\alpha)}{\alpha}$$

Algorithm Speed

- Average speed is operations per second
- Each operation requires an operator and input data
- Infinitely capable FPGA, infinite operations, finite incoming data bandwidth

Computational Speed

$$\sigma(\mu,\beta,\lambda) = \rho(\mu) * \beta * \frac{1}{1+\frac{\beta\lambda}{\mu}}$$

σ is the upper limit of computational speed in terms of computations per second

Information about the interface layer and local store in μ, λ, and β

Information about the algorithm/code in the function ρ

The fraction βλ/μ gives the relative contribution of latency to transfers

This upper speed limited is calculated separately for each layer in the memory hierarchy represented by μ, λ, β

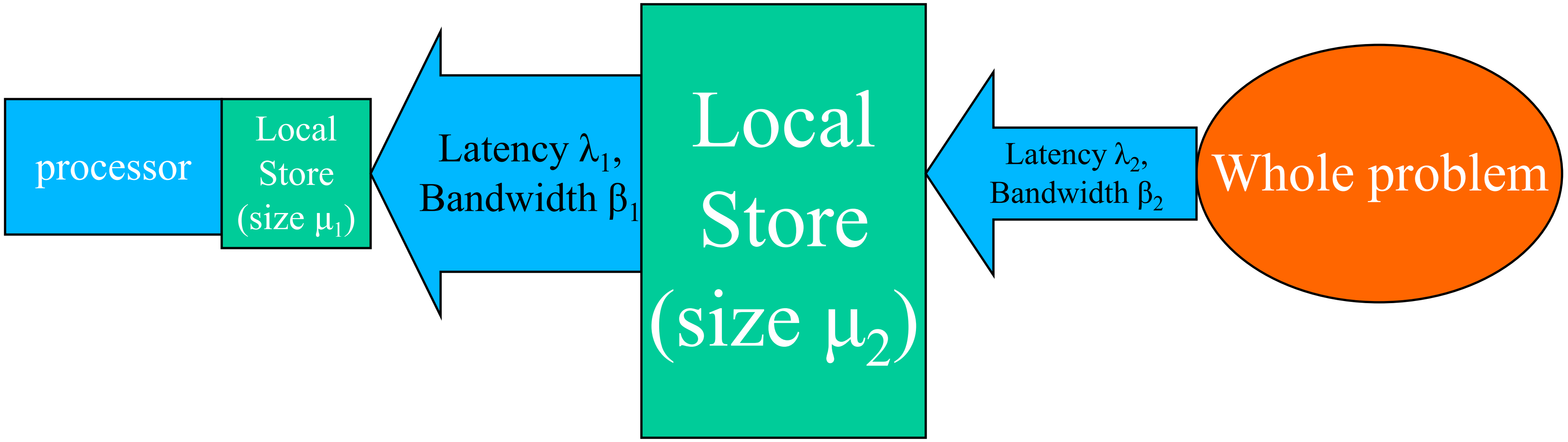
Computational Speed

$$\sigma(\mu,\beta,\lambda) = \rho(\mu) * \beta$$

If βλ/μ<<1 (the latency is a small contribution to memory transfers) then the above equation is the speed upper bound for that algorithm for that layer in the memory hierarchy

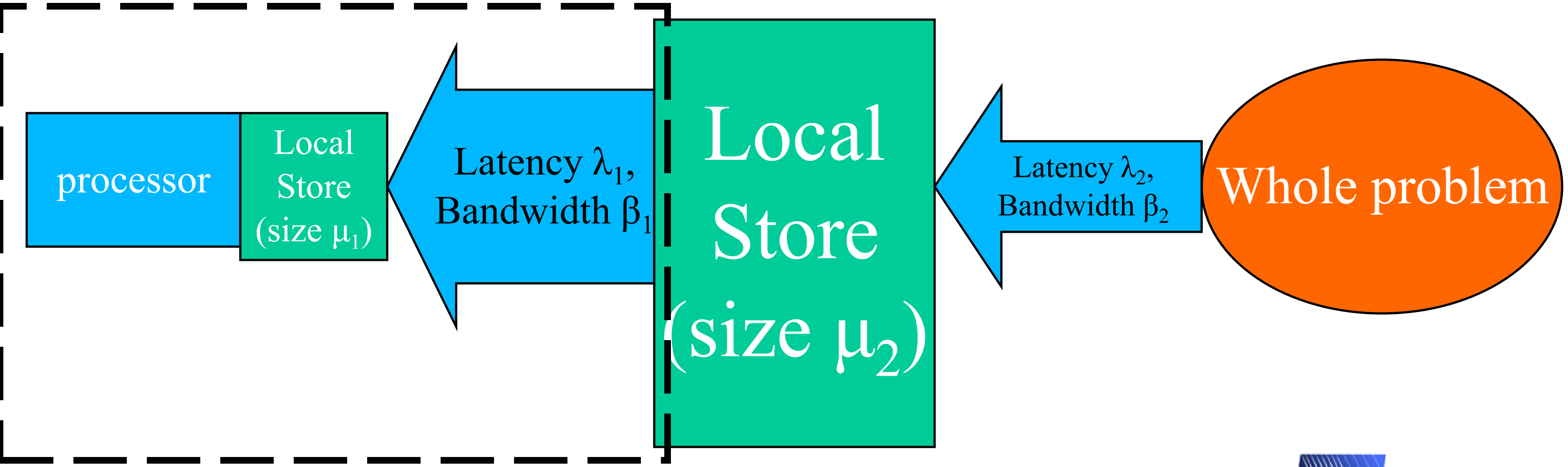
Measures Upper Performance Bound

- **There are multiple layers in each memory hierarchy, each with a local store (cache) fed by a different bus**
- **Each potentially could be the bottleneck, the depending on how the parameters of that layer interact with the computational density function at that local store size**


processor
Local Store (size μ_1)
Latency λ_1, Bandwidth β_1
Local Store (size μ_2)
Latency λ_2, Bandwidth β_2
Whole problem

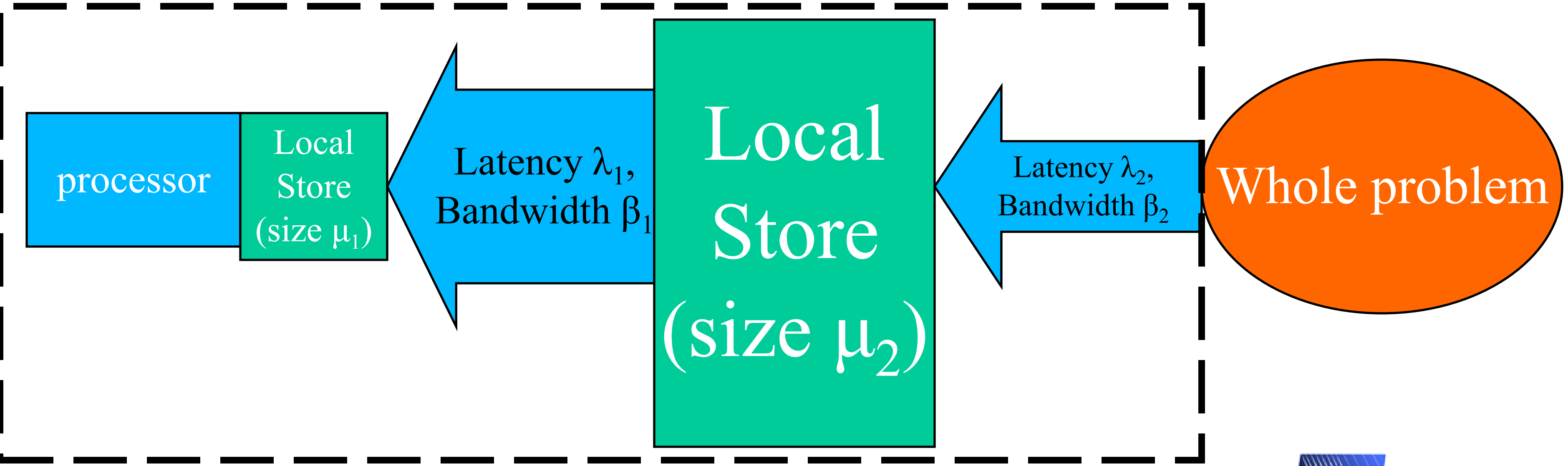
Measures Upper Performance Bound

- There are multiple layers in each memory hierarchy, each with a local store (cache) fed by a different bus
- Each potentially could be the bottleneck, the depending on how the parameters of that layer interact with the computational density function at that local store size


processor
Local Store (size μ_1)
Latency λ_1, Bandwidth β_1
Local Store (size μ_2)
Latency λ_2, Bandwidth β_2
Whole problem

Measures Upper Performance Bound

- There are multiple layers in each memory hierarchy, each with a local store (cache) fed by a different bus
- Each potentially could be the bottleneck, the depending on how the parameters of that layer interact with the computational density function at that local store size


processor
Local Store (size μ_1)
Latency λ_1, Bandwidth β_1
Local Store (size μ_2)
Latency λ_2, Bandwidth β_2
Whole problem

Data Interdependence

- Definition of computational density assumes that data is homogeneous and that multiple steps can be transferred at once; no dependencies on previous steps for in put data
- If that is *broken,* then the η(α) function does not increase past one iteration and the ρ(α) *decreases* with increasing α

Simple Example Algorithm A

- Linear element-wise vector multiply (dot product)

```
float A[SIZE],B[SIZE],C[SIZE];
for(i=0;i<size;i++){
  C[i]=A[i]*B[i];
}
```

Algorithm A Computational Density

One operation per two input parameters of size s, linear in local store size:

$$\eta(\alpha) = \frac{\alpha}{2s}$$

Computational density function is constant, not a function of the local store size. This is a purely streaming function

$$\rho(\alpha) = \frac{\frac{\alpha}{2s}}{\alpha} = \frac{1}{2s}$$

Simple Example Algorithm B

- Square Matrix-matrix multiply

```
float A[DIM][DIM],B[DIM][DIM],C[DIM][DIM];
for(i=0;i<DIM;i++){
   for(j=0;j<DIM;j++){
      C[i][j]=0.0;
      for(k=0;k<DIM;k++){
         C[i][j]+=A[i][k]*B[k][j];
      }
   }
}
```

Algorithm B Computational Density

Space required to hold 2 square matrix operands is $2sN^2$

$$\alpha = 2sN^2$$

NxN matrix multiply contains N^3 computations:

$$N = \sqrt{\frac{\alpha}{2s}}$$

Computational density:

$$\eta(\alpha) = N^3 = \left(\frac{\alpha}{2s}\right)^{3/2}$$

$$\rho(\alpha) = \frac{\left(\frac{\alpha}{2s}\right)^{3/2}}{\alpha} = \frac{\sqrt{\alpha}}{(2s)^{3/2}}$$

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

NCSA
National Center for Supercomputing Applications

Simple Example Algorithm C

- NxN interaction problem
- All inputs N particles interact with each other

$$\eta = \frac{N^2}{2} - N \approx \frac{N^2}{2}$$

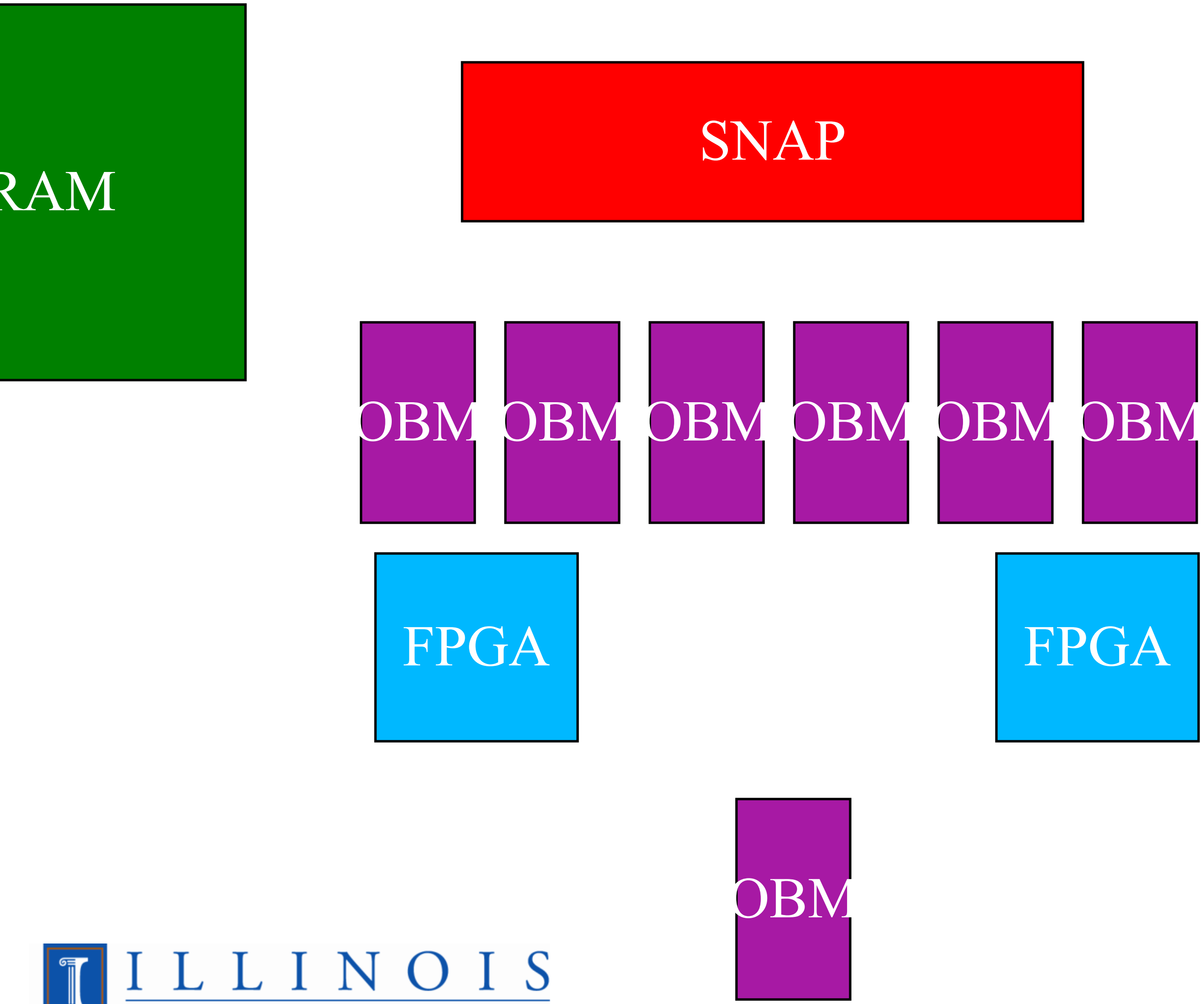
$$\alpha = N * s$$

$$N = \frac{\alpha}{s}$$

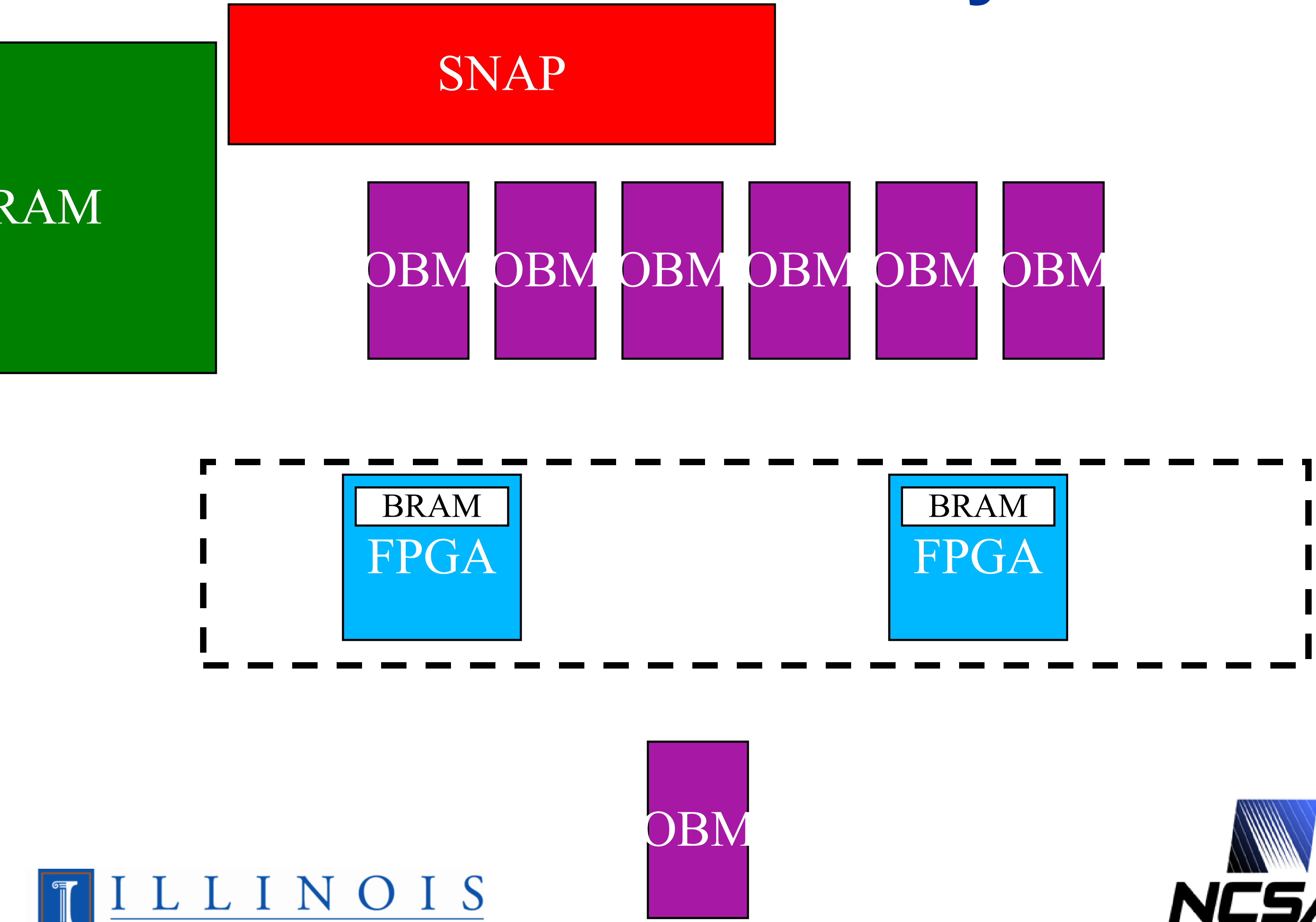
$$\eta = \frac{\left(\frac{\alpha}{s}\right)^2}{2} = \frac{\alpha^2}{2s^2}$$

$$\rho(\alpha) = \frac{\alpha}{2s^2}$$

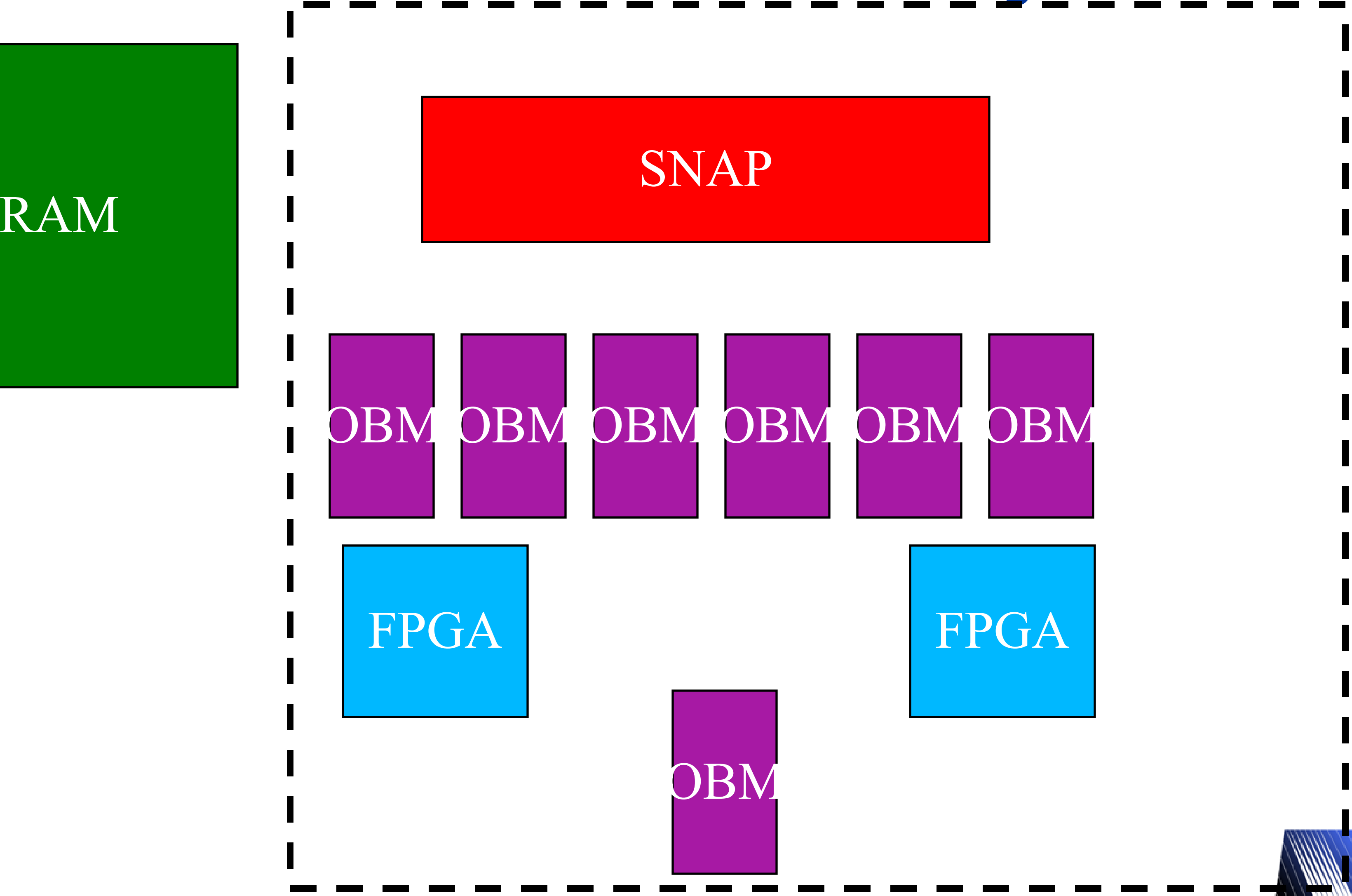
SRC MAP-C



SRC MAP-C Layer 1



SRC MAP-C Layer 2



SRC MAP-C Parameters

$\lambda_1 = 0$	$\mu_1 = .6$ MB	$\beta_1 = 6.4$ GB/s
$\lambda_2 = 20\mu s$	$\mu_2 = 28$ MB	$\beta_2 = 1.4$ GB/s

Thanks to SRC for the use of these numbers

Algorithm A on MAP-C

$$\sigma_{\mathrm{A},1} = \frac{1}{2s} \cdot \beta_1 = \frac{1}{2(4B)} \cdot (6.4GB/s) = 0.8\frac{Gops}{s}$$

$$\sigma_{\mathrm{A},2} = \frac{1}{2(4B)} \cdot (1.4GB/s) = .46\frac{Gops}{s}$$

Algorithm B on MAP-C

$$\sigma_{\mathrm{B},1} = \rho_B(.6MB)\cdot(6.4GB/s) = 219\frac{Gops}{s}$$

$$\sigma_{\mathrm{B},2} = \rho_B(28MB)\cdot(1.4GB/s) = 303\frac{Gops}{s}$$

Algorithm C on MAP-C

$$\sigma_{C,1} = \rho_C(\mu_1)\cdot\beta_1 = \rho_C(.6MB)\cdot(6.4GB/s) = 1.88\frac{Tops}{s}$$

$$\sigma_{C,2} = \rho_C(\mu_2)\cdot\beta_2 = \rho_C(28MB)\cdot(1.4GB/s) = 19.1\frac{Tops}{s}$$

What Next?

- **More specific examples**
  - NAMD
  - MILC
- **How does changing the code change the computational density?**
  - Compacting data raises graph
  - Changing loop indexing changes step positions