EE382V: Principles in Computer Architecture
Parallelism and Locality
Fall 2008

Lecture 23 – Programming the Cell BE (III) + Sequoia Compilation, Tuning, and Runtime

Mattan Erez

UT ECE

The University of Texas at Austin

Outline

- Sequoia Summary
- Other Cell programming tools
- Sequoia runtime and compilation

- Sequoia part courtesy Kayvon Fatahalian, Stanford

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Emerging Themes

- Writing high-performance code amounts to…
  - Intelligently structuring algorithms
    [compiler help unlikely]
  - Efficiently using communication
  - Efficiently using parallel resources
    [compilers struggle without help]
  - Generating efficient inner loops (kernels)
    [compilers coming around]

Sequoia

- Language: stream programming for machines with deep memory hierarchies
- Idea: Expose abstract memory hierarchy to programmer
- Implementation: **language, compiler, tuner, and runtime**
  - benchmarks run well on Cell processor based systems, clusters of PCs, SMPs, out-of-core computation, and combinations of above

Sequoia's method

- Explicit communication between abstract memories
- Locality awareness
- Hierarchy portability
  - Across machines, within levels of a machine
- **Programmer expresses combined computation and decomposition parameterized algorithm**
  - **System follows algorithm to map to a specific machine**

Sequoia summary

- Problem:
  - Deep memory hierarchies pose perf. programming challenge
  - Memory hierarchy different for different machines
- Solution: Abstract hierarchical memory in programming model
  - Program the memory hierarchy explicitly
  - Expose properties that effect performance
- Approach: Express hierarchies of tasks
  - Execute in local address space
  - Call-by-value-result semantics exposes communication
  - Parameterized for portability

Sequoia and Cell Programming Challenges

- Sequoia manages threading and synchronization
- Sequoia manages communication and all DMAs
  - Including padding and performance, but not alignment
- Sequoia manages LS
  - Allocation and packing
- Sequoia manages scheduling
  - SWP of mappar to hide communication latency
- Sequoia doesn't help with SPE code
  - Use low-level compiler tools such as XLC
- Sequoia doesn't currently help with some memory restrictions
  - Alignment, limited support but cannot rely on Sequoia
  - Banks
  - Sequoia does pad for access granularity restrictions

Outline

- Sequoia Summary
- Other Cell programming tools
- Sequoia runtime and compilation
- Sequoia part courtesy Kayvon Fatahalian, Stanford
- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Tools From IBM

- Cell SDK 3.0
  - API calls for handling communication, synchronization, and DMA
  - LIBSPU and SPUFS for getting the SPEs to do something and setting up threads and memory
  - Intrinsics for programming the SPE pipeline directly
  - GCC port for PPE and SPE part (separate compilers)
    - Only handles non-SPE specific optimizations + intrinsics
  - XLC port for PPE and SPE part (separate compilers)
    - XLC supposed to optimize for SPE pipeline with branch hints, scheduling, instruction prefetch, …
    - Automatic SIMD-ization?
- Accelerated Library Framework (ALF)
  - APIs for work queue based model to program control-plane
- "Octopiler" – single-source XLC for Cell
  - OpenMP directives
  - Relies on SW cache to get the OpenMP working
  - Automatic SIMD-ization

Tools from Industry

- Mercury Systems
  - Array based language
  - Highly-tuned BLAS and FFT
- RapidMind
  - Dynamically compiled program
  - Relies on array data types
  - Builds up kernels and DMAs

Tools From Academia

- Sequoia
- Cell Superscalar (CellSs)
  - Program with OpenMP like directives to identify kernels
  - Uses SW cache intensively
  - Runtime applies superscalar style optimization and scheduling to coarse-grained kernels (identified above)
- Charm++
  - Runtime based approach
  - Objects with explicit communication and "entry points" for synchronization
  - Uses a work queue and peaks into it to do the DMAs

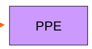
Outline

- Sequoia Summary
- Other Cell programming tools
- Sequoia runtime and compilation
- Sequoia part courtesy Kayvon Fatahalian, Stanford
- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

How this all works (Cell example)

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

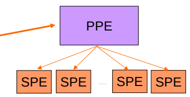
```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2D(...);
    C = sqAlloc2D(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```



How this all works

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2D(...);
    C = sqAlloc2D(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```



PPE launches bootstrap threads on SPEs

How this all works

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2D(...);
    C = sqAlloc2D(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

Allocate data

How this all works

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2D(...);
    C = sqAlloc2D(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

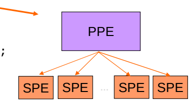
Call task

Top level task call

```
task matmul::inner(in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N])
{
  tunable int P, Q, R;

  mappar( int i=0 to M/P,
          int j=0 to N/R) {
     mapseq( int k=0 to T/Q ) {

        matmul(A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
               B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
               C[P*i:P*(i+1);P][R*j:R*(j+1);R]);
     }
  }
}
```



PPE mails SPE leaf task to instruct olay load and execution

Leaf task call

```
task matmul::inner(in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N])
{
  tunable int P, Q, R;

  mappar( int i=0 to M/P,
          int j=0 to N/R) {
     mapseq( int k=0 to T/Q ) {

        matmul(A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
               B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
               C[P*i:P*(i+1);P][R*j:R*(j+1);R]);
     }
  }
}
```

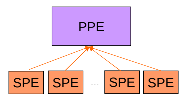
SPE id controls assignment of iteration space and DMA list offsets

Leaf task return

```
task matmul::inner(in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N])
{
  tunable int P, Q, R;

  mappar( int i=0 to M/P,
          int j=0 to N/R) {
     mapseq( int k=0 to T/Q ) {

        matmul(A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
               B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
               C[P*i:P*(i+1);P][R*j:R*(j+1);R]);
     }
  }
}
```



SPE mails PPE and waits for command

Control return to user code

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

Kill off threads and cleanup

Autotuning Sequoia Programs

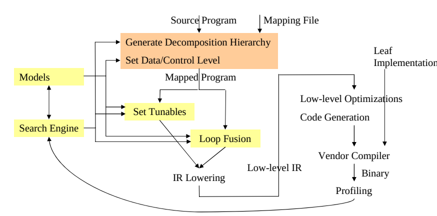
- Autotuner helps user with mapping (user can always override)

Specialization with Autotuning

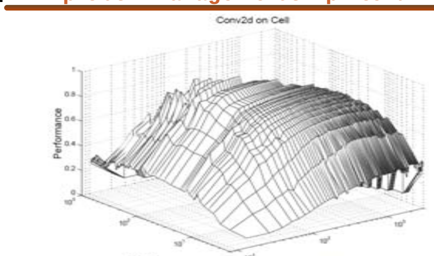
- Work by Manman Ren (Stanford), PACT 2008
- Use Sequoia to identify what needs tuning
  - Explicit tunables and parameters in the language
- Tuning framework for SW-managed hierarchies
- Automatic profile guided search across tunables
  - Aggressive pruning
  - Illegal parameters (don't fit in memory level)
  - Tunable groups
  - Programmer input on ranges
  - Coarse → fine search
- Loop fusion across multiple loop levels
  - Measure profitability from tunable search
  - Adjust for "tunable mismatch"
  - Realign reuse to reduce communication

Overview: mapping the program

- Mapped versions are generated
  - Matching the decomposition hierarchy with the machine hierarchy
  - Choosing a variant for each call site
  - Set level of data objects and control statements



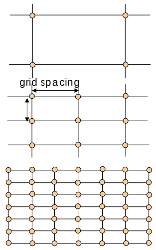
Explicit SW Management Simplifies Tuning



- Smooth search space
- Performance models can also work
  - For Cell, not cluster

The search algorithm

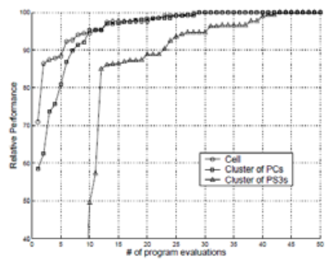
- A pyramid search
- A greedy search algorithm at each grid level
  - Achieve good performance quickly due to smoothness of the search space



Start with a coarse grid
Refine the grid when
no further progress can
be made

Guided Search Converges Quickly

- Smoothness leads to quick convergence



Autotuning Out Performs Programmer

		CONV2D	SGEMM	FFT3D	SUmb
Cell	**auto** hand	**99.6** 85	**137** 119	**57** 54	**12.1**
Cluster of PCs	**auto** hand	**26.7** 24	**92.4** 90	**5.5** 5.5	**2.2**
Cluster of PS3s	**auto** hand	**20.7** 19	**33.4** 30	**0.57** 0.36	**0.63**

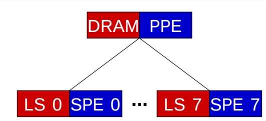
Sequoia compilation

- A compiler for hierarchical bulk operations

Compiler for hierarchical bulk operations

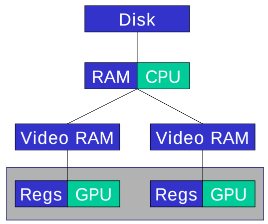
1. Is built around the portable abstractions of hierarchical memory and bulk operations.
2. Automatically manages all program data movement to increase memory throughput.
3. Automatically allocates explicitly managed memories.

We abstract Cell as a tree of memories

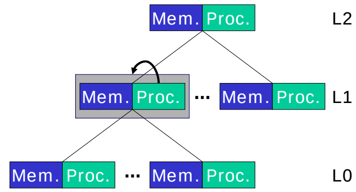


- Why a tree of memories?
- A portable abstraction that matches our target machines of interest: Cell, clusters, etc.

- We can model other machines as trees of memories (e.g.) a dual-GPU machine backed by disk storage:
- Not all tree nodes have a processor.
- Leaves of tree are compute-intensive
- processors.

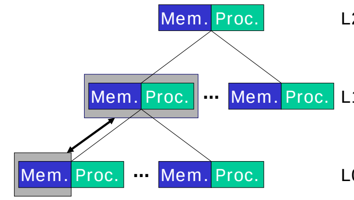


Modeled machine capabilities



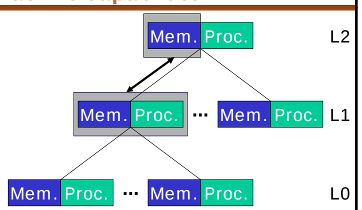
- 1. Execute out of its local memory.

Modeled machine capabilities



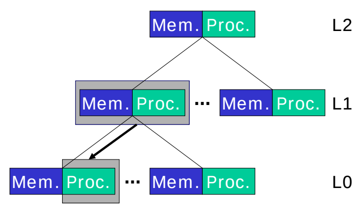
- 1. Execute out of its local memory.
- 2. Transfer data to/from a child.

Modeled machine capabilities



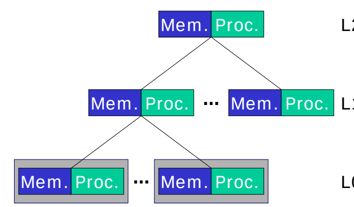
- 1. Execute out of its local memory.
- 2. Transfer data to/from a child.
- 3. Transfer data to/from its parent.

Modeled machine capabilities



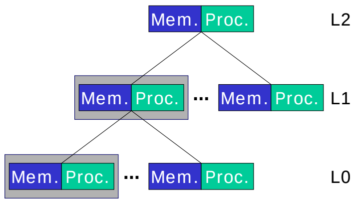
- 1. Execute out of its local memory.
- 2. Transfer data to/from a child.
- 3. Transfer data to/from its parent.
- 4. Launch code in a child.

Parallelism in the abstract machine model



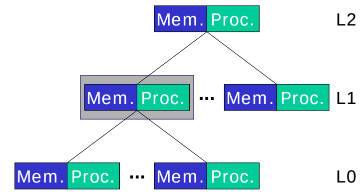
- 1. Parallel PEs within a level.

Parallelism in the abstract machine model



- 1. Parallel PEs within a level.
- 2. Concurrent parent/child execution.

Parallelism in the abstract machine model



- 1. Parallel PEs within a level.
- 2. Concurrent parent/child execution.
- 3. Parallel execution within a PE.

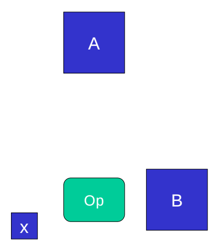
Modeling Programs

Programs comprise four elements

-
- 1. Operations (blue).

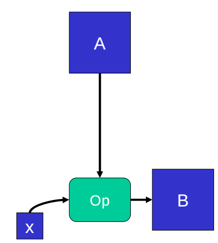
Programs comprise four elements

- 1. Operations (blue).
- 2. Data (red).



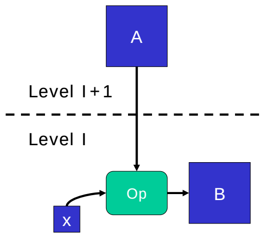
Programs comprise four elements

- 1. Operations (blue).
- 2. Data (red).
- 3. Dependences.



Programs comprise four elements

- 1. Operations (green).
- 2. Data (blue).
- 3. Dependences.
- 4. Machine levels.



Programs comprise four elements

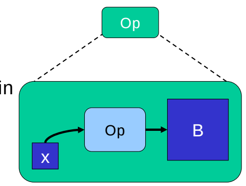
- 1. Operations (green).
- 2. Data (blue).
- 3. Dependences.
- 4. Machine levels.

- Operations may contain nested subprograms.

Programs comprise four elements

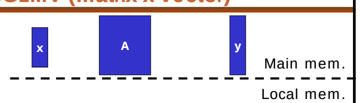
- 1. Operations (blue).
- 2. Data (red).
- 3. Dependences.
- 4. Machine levels.

- Operations may contain nested subprograms.



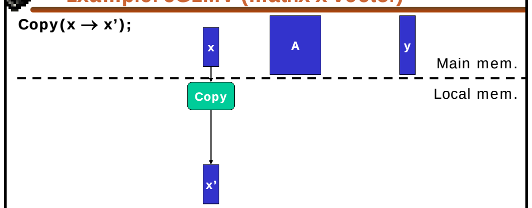
Example: SGEMV (matrix x vector)

y = A . x



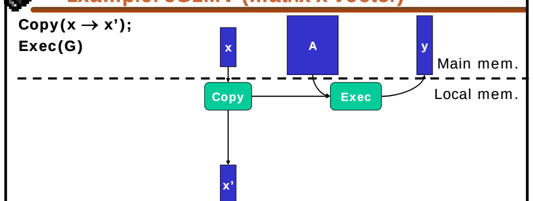
Example: SGEMV (matrix x vector)

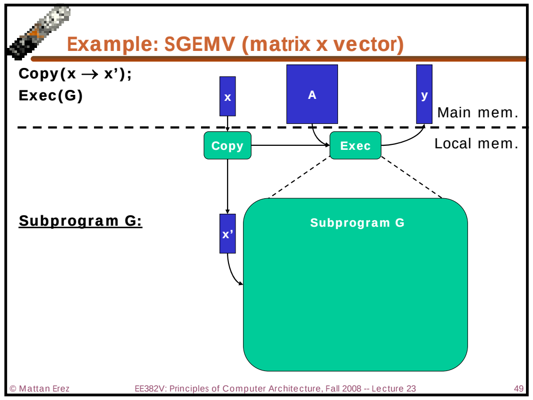
Copy(x → x');

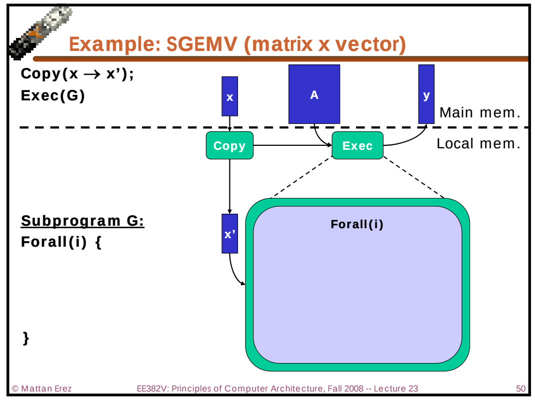


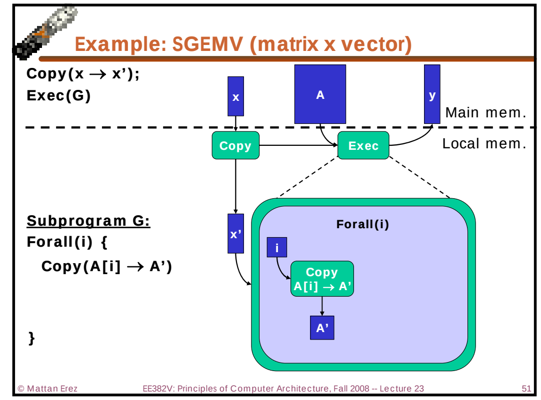
Example: SGEMV (matrix x vector)

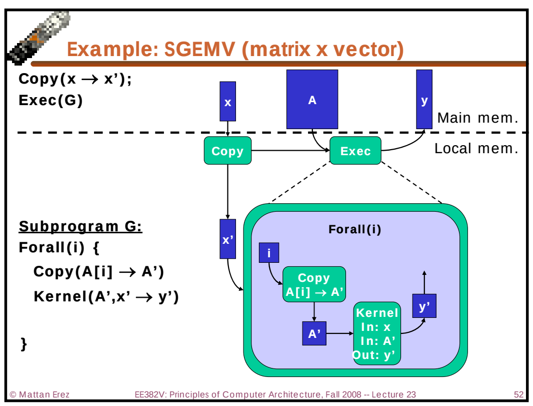
Copy(x → x');
Exec(G)

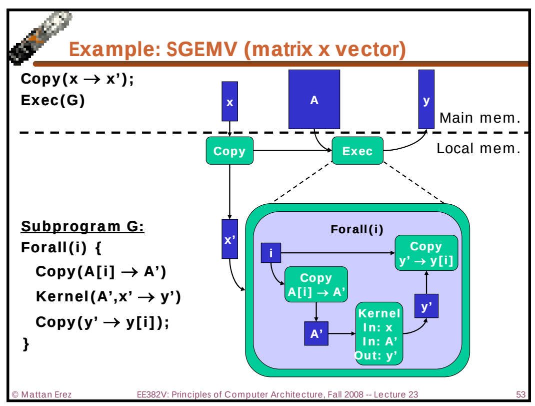


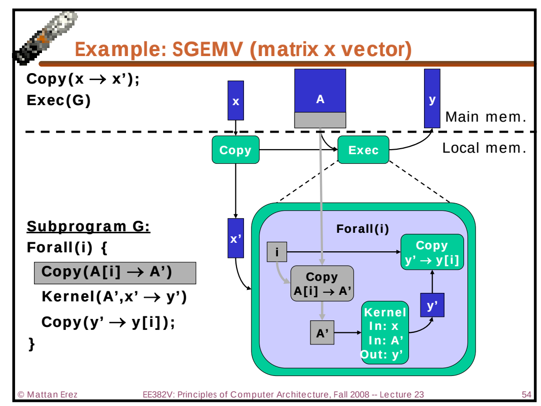

Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
x
A
y
Copy
Exec
x'
Subprogram G
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
49


Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
Forall(i) {
}
x
A
y
Copy
Exec
x'
Forall(i)
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
50


Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
Forall(i) {
Copy(A[i] → A')
}
x
A
y
Copy
Exec
x'
Forall(i)
i
Copy
A[i] → A'
A'
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
51


Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
Forall(i) {
Copy(A[i] → A')
Kernel(A',x' → y')
}
x
A
y
Copy
Exec
x'
Forall(i)
i
Copy
A[i] → A'
A'
Kernel
In: x
In: A'
Out: y'
y'
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
52


Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
Forall(i) {
Copy(A[i] → A')
Kernel(A',x' → y')
Copy(y' → y[i]);
}
x
A
y
Copy
Exec
x'
Forall(i)
i
Copy
A[i] → A'
A'
Kernel
In: x
In: A'
Out: y'
y'
Copy
y' → y[i]
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
53

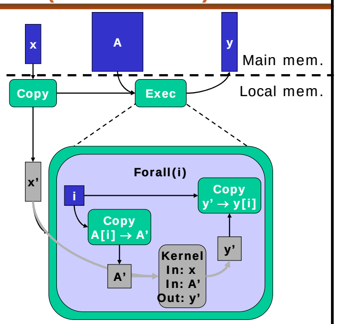

Example: SGEMV (matrix x vector)
Copy(x → x');
Exec(G)
Main mem.
Local mem.
Subprogram G:
Forall(i) {
Copy(A[i] → A')
Kernel(A',x' → y')
Copy(y' → y[i]);
}
x
A
y
Copy
Exec
x'
Forall(i)
i
Copy
A[i] → A'
A'
Kernel
In: x
In: A'
Out: y'
y'
Copy
y' → y[i]
© Mattan Erez
EE382V: Principles of Computer Architecture, Fall 2008 -- Lecture 23
54

Example: SGEMV (matrix x vector)

```
Copy(x → x');
Exec(G)
```

Subprogram G:

```
Forall(i) {
  Copy(A[i] → A')
  Kernel(A',x' → y')
  Copy(y' → y[i]);
}
```

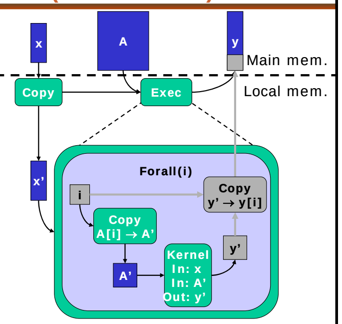


Example: SGEMV (matrix x vector)

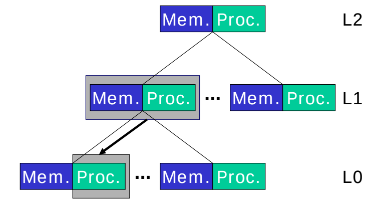
```
Copy(x → x');
Exec(G)
```

Subprogram G:

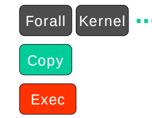
```
Forall(i) {
  Copy(A[i] → A')
  Kernel(A',x' → y')
  Copy(y' → y[i]);
}
```



Modeled machine capabilities



- 1. **Execute out of its local memory.**
- **2. Transfer data to/from a child.**
- **3. Transfer data to/from its parent.**
- **4. Launch code in a child.**



Program IR properties:

1. Is mechanism-independent.
2. Features scalar and bulk operations within a single framework.
3. Spans the entire machine (all levels).
4. Fully captures all program semantics.
5. Exposes parallelism via dependences and "Forall" operations.

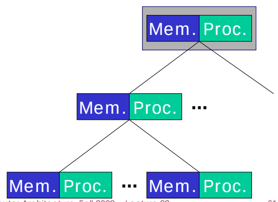
Optimizing Programs

Optimizations are IR transformations

- They can be applied at any hierarchy level of the program or the machine.

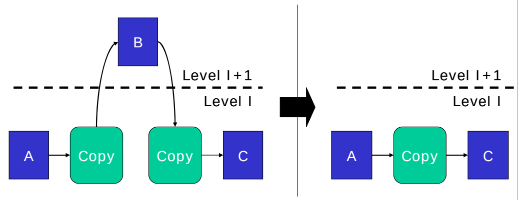
Optimizations are IR transformations

- They can be applied at any hierarchy level of the program or the machine.
- Correctness: Transformations preserve top-level program inputs and outputs.



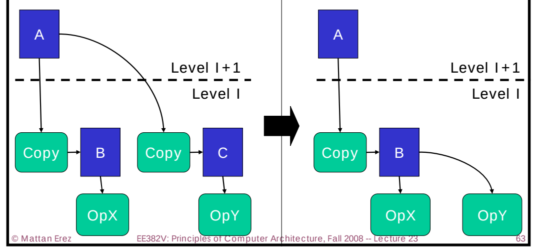
Optimization: Copy elimination

- 1. Removing spills [producer-consumer locality].

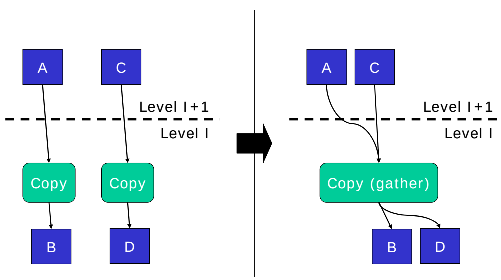


Optimization: Copy elimination

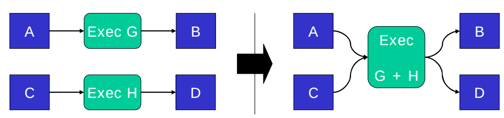
- 1. Removing spills [producer-consume locality].
- 2. Removing duplicates [temporal locality.].



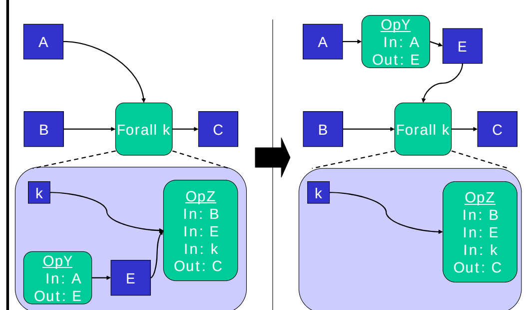
Optimization: Copy grouping



Optimization: Exec grouping



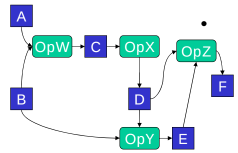
Optimization: Hoisting



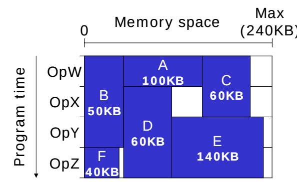
Scheduling optimizations

1. Software pipelining.
2. Competing heuristics:
   1. Maximize operation concurrency.
   2. Group similar operations (e.g. Execs) to amortize issue overheads.

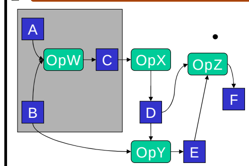
Optimization: Space allocation



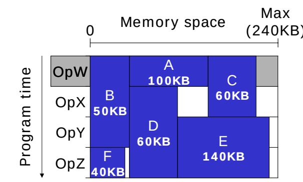
Global view of data usage: better than LRU.



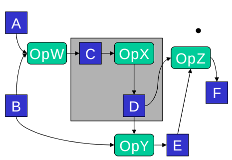
Optimization: Space allocation



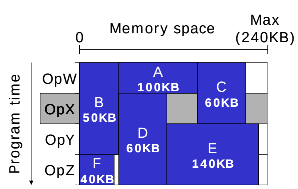
Global view of data usage: better than LRU.



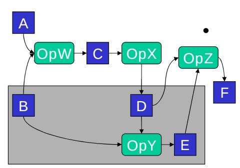
Optimization: Space allocation



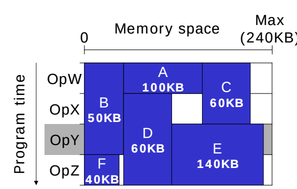
Global view of data usage: better than LRU.



Optimization: Space allocation

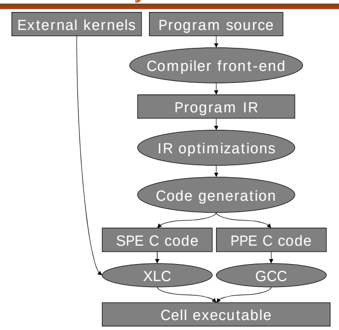


Global view of data usage: better than LRU.



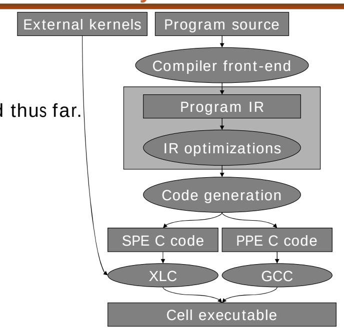
Targeting a Cell Processor

An overview of our system



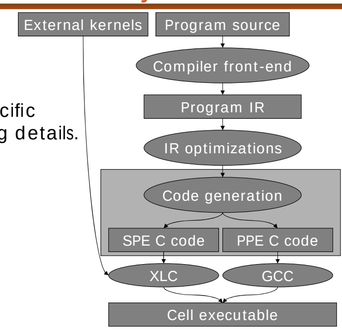
An overview of our system

- Covered thus far.

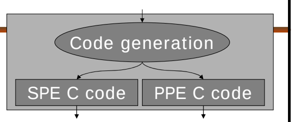


An overview of our system

- Cell-specific mapping details.



A few details



1. Our source-to-source compiler generates two sets of output files.
2. Each Exec operation → SPE overlay.
3. Each Copy operation → DMA command.
   - Data objects and transfers padded to multiples of 16 bytes.
4. DMAs and SPE kernels overlapped where possible.

Summary

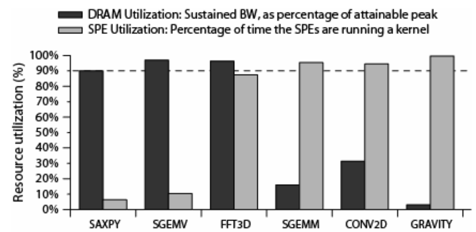
Summary of results

- Performance competitive with native code
  - Generic optimization for hierarchical bulk programs
    - Copy elimination
    - DMA transfer coalescing
    - Operation hoisting
    - Array allocation / packing
    - Scheduling (tasks and DMAs)
  - Automatic tuning for performance
- Portable: no source-code changes for different configurations
  - Cell, SMP, Cluster, Disk
  - Compositions of above
  - Automatic tuning
- Maximizes resources (compute or communication)
- Low overhead

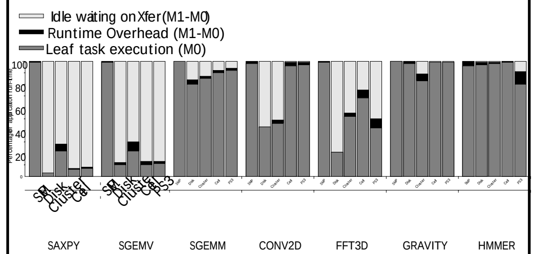
Results – Horizontal portability - GFlop/s

	Scalar	SMP	Disk	Cluster	Cell	PS3
SAXPY	0.3	**0.7**	0.007	1.4	**3.5**	3.1
SGEMV	1.1	**1.7**	0.04	3.8	**12**	10
SGEMM	6.9	**45**	5.5	91	**119**	94
CONV2D	1.9	**7.8**	0.6	24	**85**	62
FFT3D	1.5	**7.8**	0.1	7.5	**54**	31*
GRAVITY	4.8	**40**	3.7	68	**97**	71
HMMER	0.9	**11**	0.9	12	**12**	7.1*

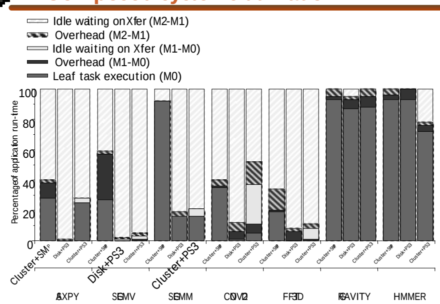
Cell utilization



2 Level Utilization

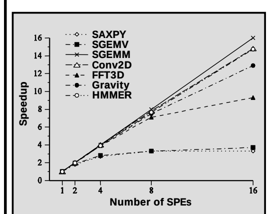


Composed systems utilization

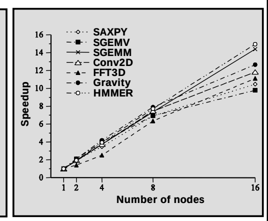


Performance scaling

SPE scaling on 2.4GHz Dual-Cell blade



Scaling on P4 cluster with Infiniband interconnect



Sequoia limitations

- Require explicit declaration of working sets
  - Programmer must know what to transfer
  - Some irregular applications present problems
- Task mapping somewhat laborious
  - Autotuning helps
  - **Understand which parts can be automated better**

Sequoia summary

- Enforce structuring already required for performance as integral part of programming model
- Make these hand optimizations portable and easier to perform

Sequoia summary (http://sequoia.stanford.edu)

- Problem:
  - Deep memory hierarchies pose perf. programming challenge
  - Memory hierarchy different for different machines
- Solution: Abstract hierarchical memory in programming model
  - Program the memory hierarchy explicitly
  - Expose properties that effect performance
- Approach: Express hierarchies of tasks
  - Execute in local address space
  - Call-by-value-result semantics exposes communication
  - Parameterized for portability