Polly Labs

Argonne NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Code Analysis and Optimization using Loop Hierarchies

Workshop on Program Synthesis for Scientific Computing

Michael Kruse, Hal Finkel

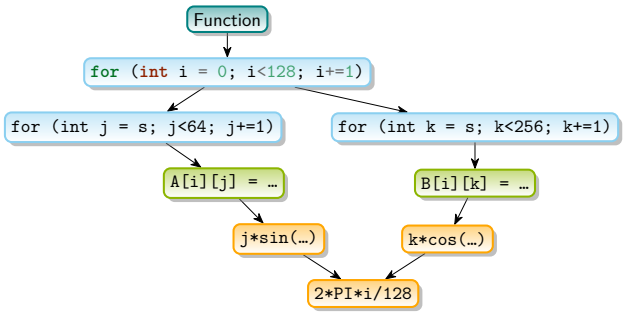
Argonne Leadership Computing Facility
Argonne National Laboratory

2020-08-05

Table of Contents

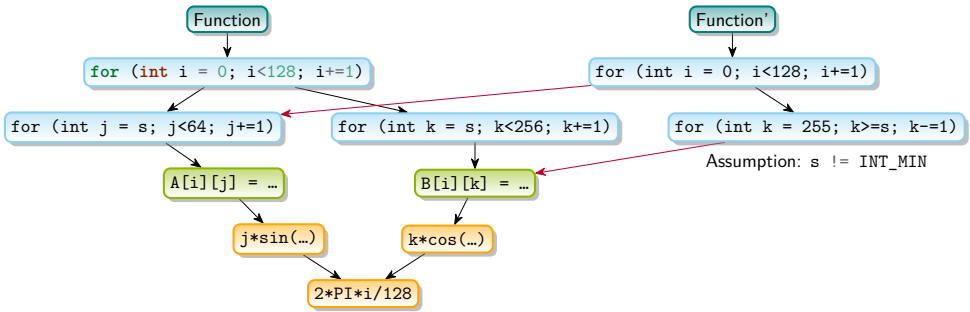
Loop Hierarchy DAG

```
void Function(int s) {
    for (int i = 0; i < 128; i+=1) {
        for (int j = s; j <  64; j+=1) A[i][j] = j*sin(2*PI*i/128);
        for (int k = s; k < 256; k+=1) B[i][k] = k*cos(2*PI*i/128);
    }
}
```

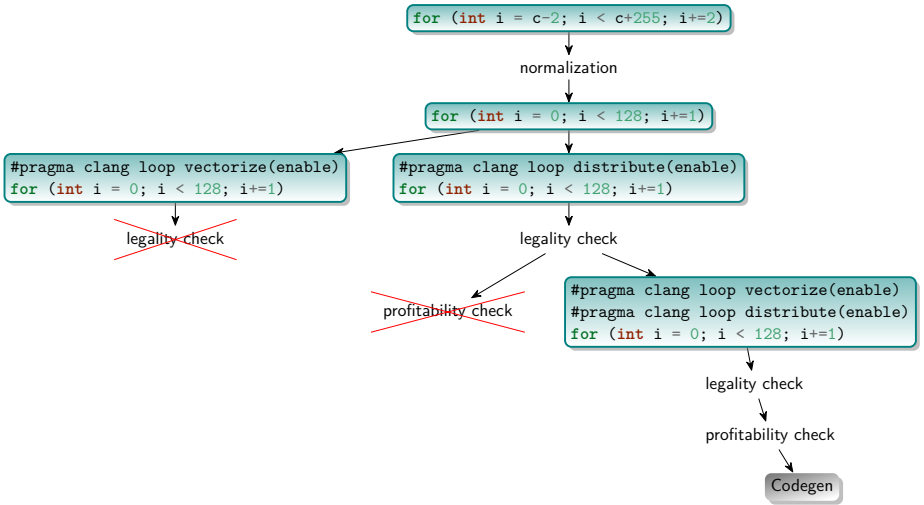


Loop Hierarchy DAG

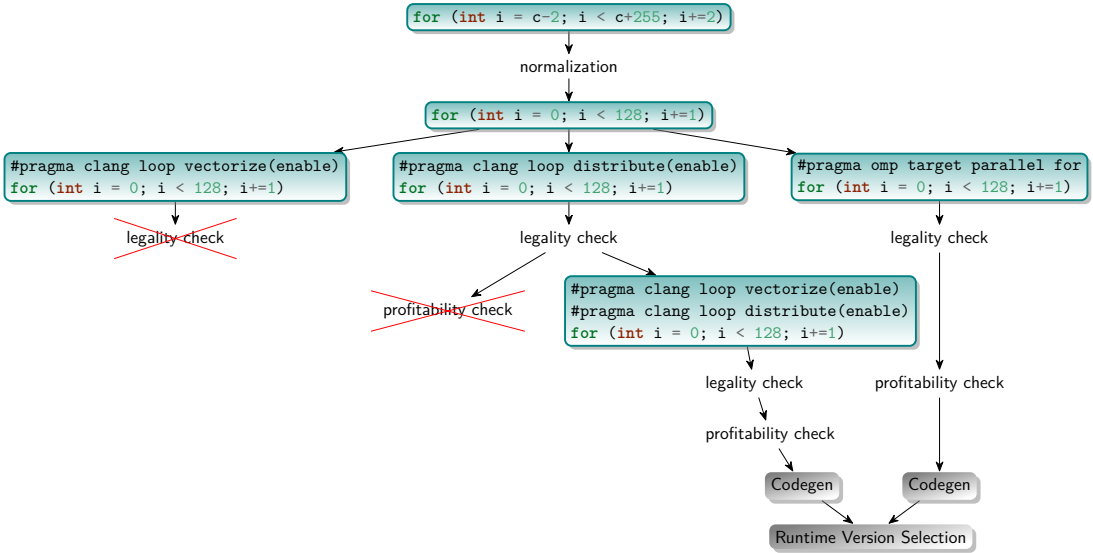
```
void Function(int s) {
    for (int i = 0; i < 128; i+=1) {
        for (int j = s  ; j <  64; j+=1) A[i][j] = j*sin(2*PI*i/128);
        for (int k = 255; k >= s ; k-=1) B[i][k] = k*cos(2*PI*i/128);
    }
}
```



Shallow Copy Mechanics



Shallow Copy Mechanics



Legality Check

Comparison between known-good (original) and transformed loop tree

- Executes all statement instances
- No additional instances
- No dependency reversals

Profitability Check

Execution Time Model

- Estimate cycles of straight-line code
  - `llvm-mca`
  - Memory access latency
- Estimate trip count
  - Arbitrary constant (“100”)
  - “infinity” (only innermost kernel counts)
  - From user annotations (`#pragma loop count(n)`)
  - From PGO / previous JIT stage

Profitability Check

Execution Time Model

- Estimate cycles of straight-line code
  - `llvm-mca`
  - Memory access latency
- Estimate trip count
  - Arbitrary constant (“100”)
  - “infinity” (only innermost kernel counts)
  - From user annotations (`#pragma loop count(n)`)
  - From PGO / previous JIT stage

Autotuning

- Select most-promising not-yet-evaluated

– or –

- Select know-fastest

Profitability Check

Execution Time Model

- Estimate cycles of straight-line code
  - `llvm-mca`
  - Memory access latency
- Estimate trip count
  - Arbitrary constant (“100”)
  - “infinity” (only innermost kernel counts)
  - From user annotations (`#pragma loop count(n)`)
  - From PGO / previous JIT stage

Autotuning

- Select most-promising not-yet-evaluated

– or –

- Select know-fastest

Machine Learning

- Apply a per-architecture pre-trained model

Static Compute Hierarchy

CPU-based

Loop structure	Compute hierarchy	
`for (int step = 0; step < STEPS; step+=1)`	Sequential	
`for (int i = 0; i < I; i+=1)`	Cluster nodes	MPI/PGAS
`for (int j = 0; j < J; j+=1)`	Processors	thread-parallelization
`for (int k = 0; k < K; k+=1)`	Cores	thread-parallelization
`for (int l = 0; l < L; l+=1)`	SMT	thread-parallelization
`for (int m = 0; m < M; m+=1)`	SIMD	vectorization
`for (int n = 0; n < N; n+=1)`	Superscalar	unrolling+instruction scheduling
`Body(i, ...)`		

Static Compute Hierarchy

CPU-based

Loop structure	Compute hierarchy		
`for (int step = 0; step < STEPS; step+=1)`	Sequential		
`for (int i = 0; i < I; i+=1)`	Cluster nodes	MPI/PGAS	Parallel
`for (int j = 0; j < J; j+=1)`	Processors	thread-parallelization	Parallel
`for (int k = 0; k < K; k+=1)`	Cores	thread-parallelization	Parallel
`for (int l = 0; l < L; l+=1)`	SMT		Concurrent
`for (int m = 0; m < M; m+=1)`	SIMD	vectorization	Lock-step
`for (int n = 0; n < N; n+=1)`	Superscalar	unrolling+instruction scheduling	Transparent
`Body(i, ...)`			

Static Compute Hierarchy

CPU-based

Loop structure	Compute hierarchy	
`for (int step = 0; step < STEPS; step+=1)`	Sequential	
`for (int i = 0; i < I; i+=1)`	Cluster nodes	MPI/PGAS
`for (int j = 0; j < J; j+=1)`	Processors	thread-parallelization
`for (int k = 0; k < K; k+=1)`	Cores	thread-parallelization
`for (int l = 0; l < L; l+=1)`	SMT	thread-parallelization
`for (int m = 0; m < M; m+=1)`	SIMD	vectorization
`for (int n = 0; n < N; n+=1)`	Superscalar	unrolling+instruction scheduling
`Body(i, ...)`		

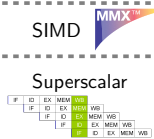
Distributed Memory

Shared Memory

Static Compute Hierarchy

CPU-based

Loop structure	Compute hierarchy	
`for (int step = 0; step < STEPS; step+=1)`	Sequential	
`for (int i = 0; i < I; i+=1)`	Cluster nodes	MPI/PGAS
`for (int j = 0; j < J; j+=1)`	Processors	Main Memory
`for (int k = 0; k < K; k+=1)`	Cores	L2-Cache
`for (int l = 0; l < L; l+=1)`	SMT	L1-Cache
`for (int m = 0; m < M; m+=1)`	SIMD	vectorization
`for (int n = 0; n < N; n+=1)`	Superscalar	unrolling+instruction scheduling
`Body(i, ...)`		



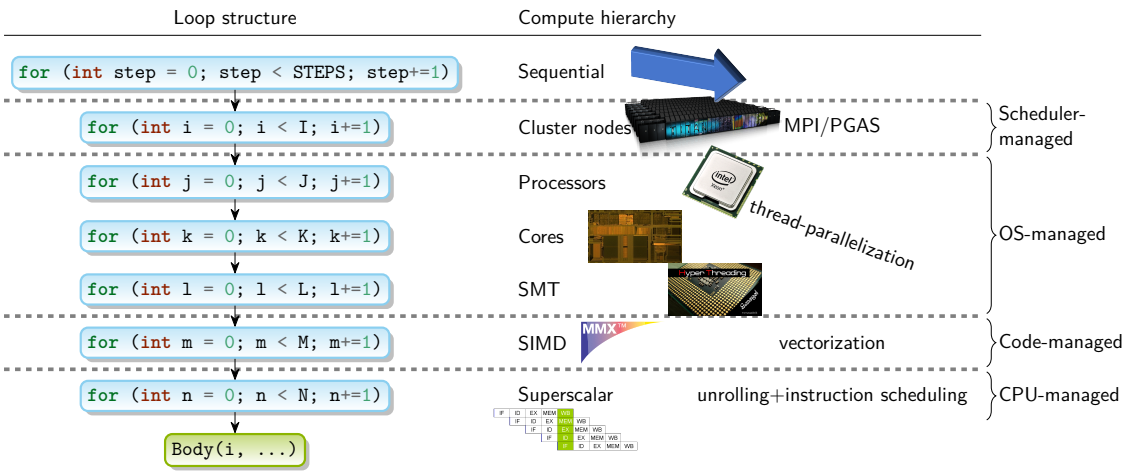
Static Compute Hierarchy

CPU-based

Loop structure	Compute hierarchy		
`for (int step = 0; step < STEPS; step+=1)`	Sequential		
`for (int i = 0; i < I; i+=1)`	Cluster nodes	MPI/PGAS	Oversubscribable
`for (int j = 0; j < J; j+=1)`	Processors	thread-parallelization	Oversubscribable
`for (int k = 0; k < K; k+=1)`	Cores	thread-parallelization	Oversubscribable
`for (int l = 0; l < L; l+=1)`	SMT	thread-parallelization	Oversubscribable
`for (int m = 0; m < M; m+=1)`	SIMD	vectorization	Hard limit
`for (int n = 0; n < N; n+=1)`	Superscalar	unrolling+instruction scheduling	Hard limit
`Body(i, ...)`			

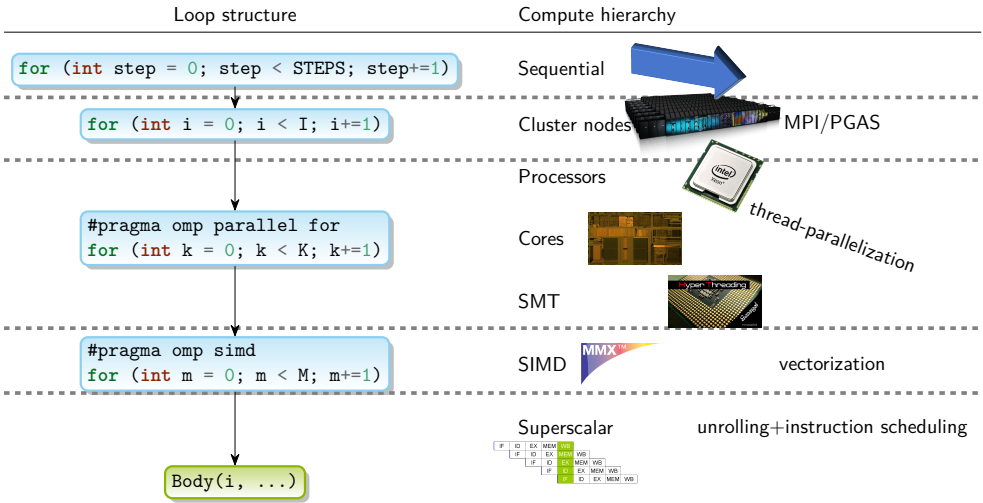
Static Compute Hierarchy

CPU-based



Static Compute Hierarchy

CPU-based (OpenMP)



Static Compute Hierarchy

GPU-based (CUDA)

Loop structure	Compute hierarchy
`for (int step = 0; step < STEPS; step+=1)`	Sequential
`for (int i = 0; i < I; i+=1)`	Cluster nodes GPUDirect
`for (int j = 0; j < J; j+=1)`	Accelerator cards offloading
`for (int k = 0; k < K; k+=1)`	Multiprocessors (Grid)
`for (int l = 0; l < L; l+=1)`	CUDA Core (Block)
`for (int m = 0; m < M; m+=1)`	SIMT (Warp)
`for (int n = 0; n < N; n+=1)`	SIMD (`f16x2`, `vadd2`, ...) vectorization
`Body(i, ...)`	

Static Compute Hierarchy

GPU-based (OpenMP-to-ptx)

Loop structure	Compute hierarchy
`for (int step = 0; step < STEPS; step+=1)`	Sequential
`for (int i = 0; i < I; i+=1)`	Cluster nodes  GPUDirect
`#pragma omp target`	Accelerator cards  offloading
`#pragma omp teams distribute` `for (int k = 0; k < K; k+=1)`	Multiprocessors (Grid) 
	CUDA Core (Block) 
`#pragma omp parallel for` `for (int l = 0; l < L; l+=1)`	SIMT (Warp) 
	SIMD (`f16x2, vadd2, ...`)  vectorization
`Body(i, ...)`	

Compute Hierarchy Mapping

Loop structure

```
for (int step = 0; step < STEPS; step+=1)
```

```
for (int i = 0; i < I; i+=1)
```

?

```
Body(i, ...)
```

Compute hierarchy

Sequential

Cluster nodes

Accelerator cards

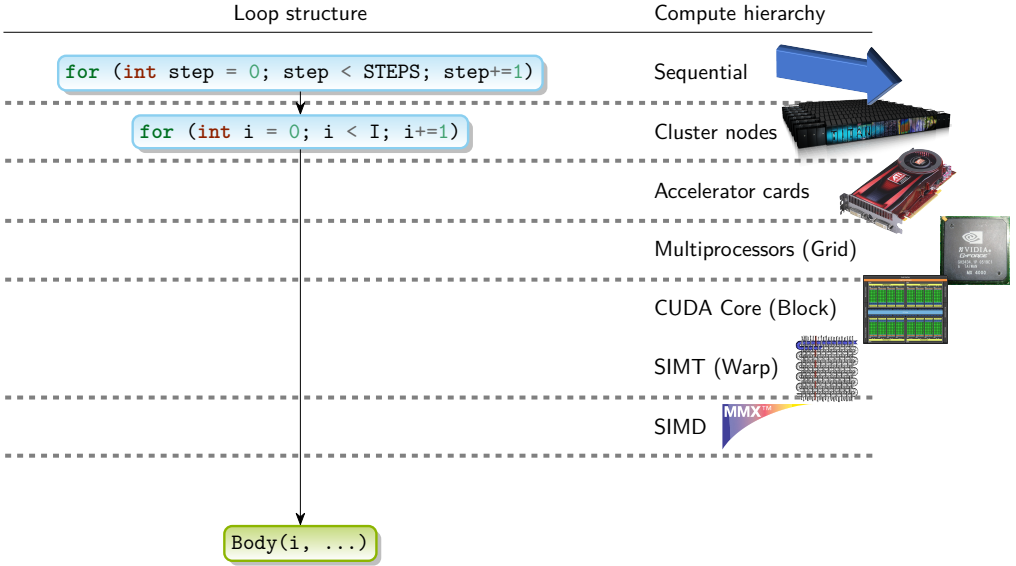
Multiprocessors (Grid)

CUDA Core (Block)

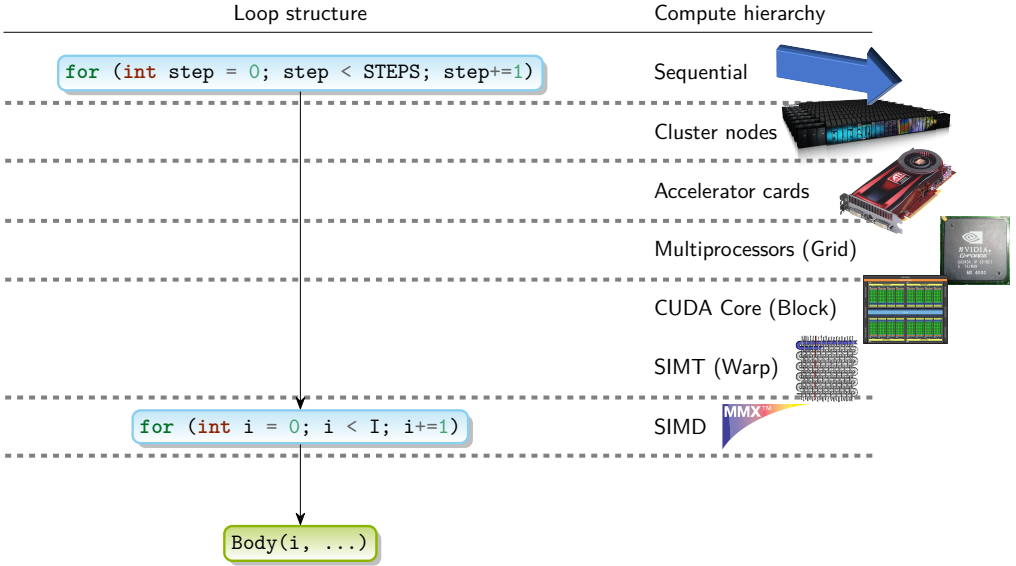
SIMT (Warp)

SIMD

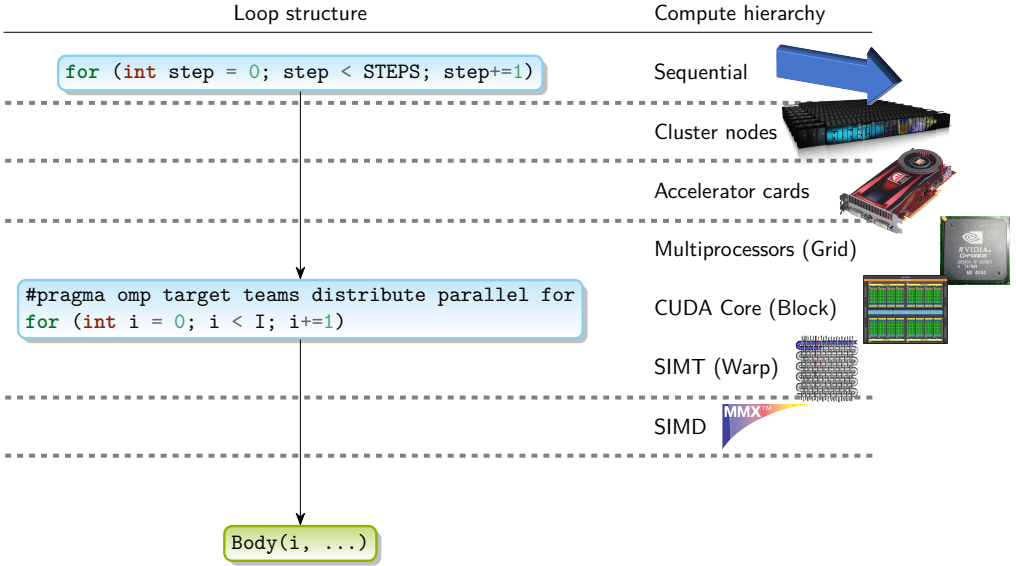
Compute Hierarchy Mapping



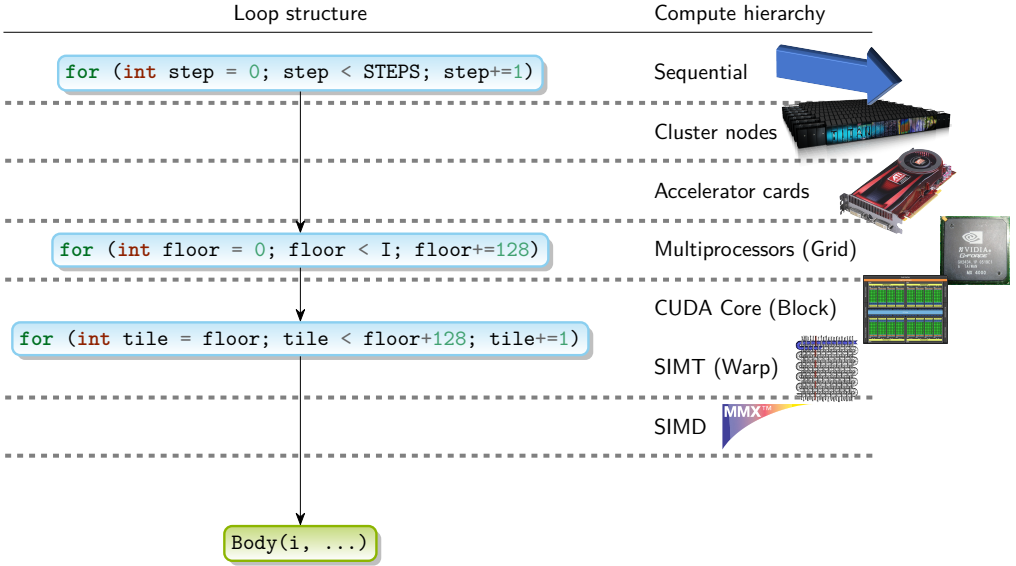
Compute Hierarchy Mapping



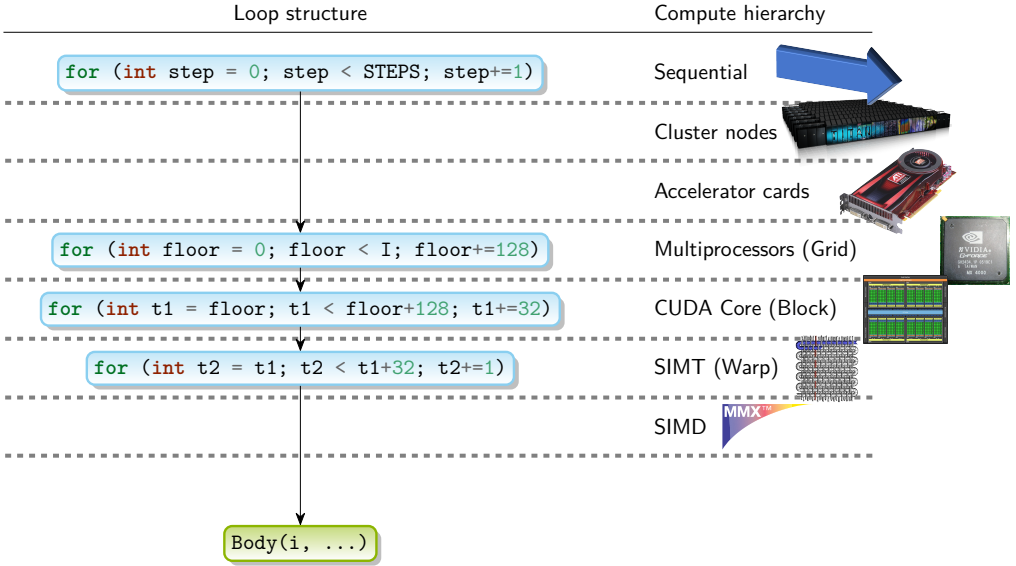
Compute Hierarchy Mapping



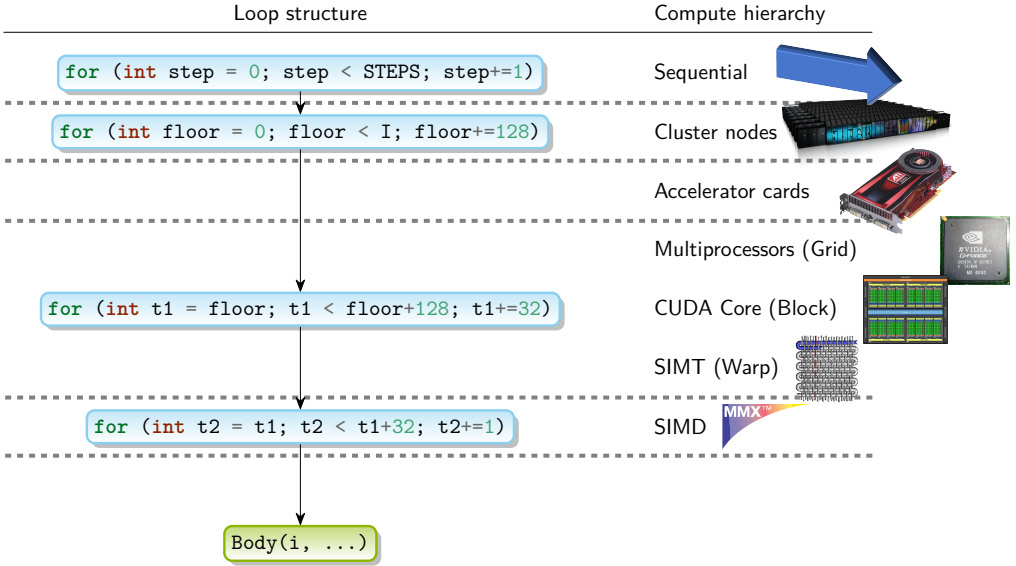
Compute Hierarchy Mapping



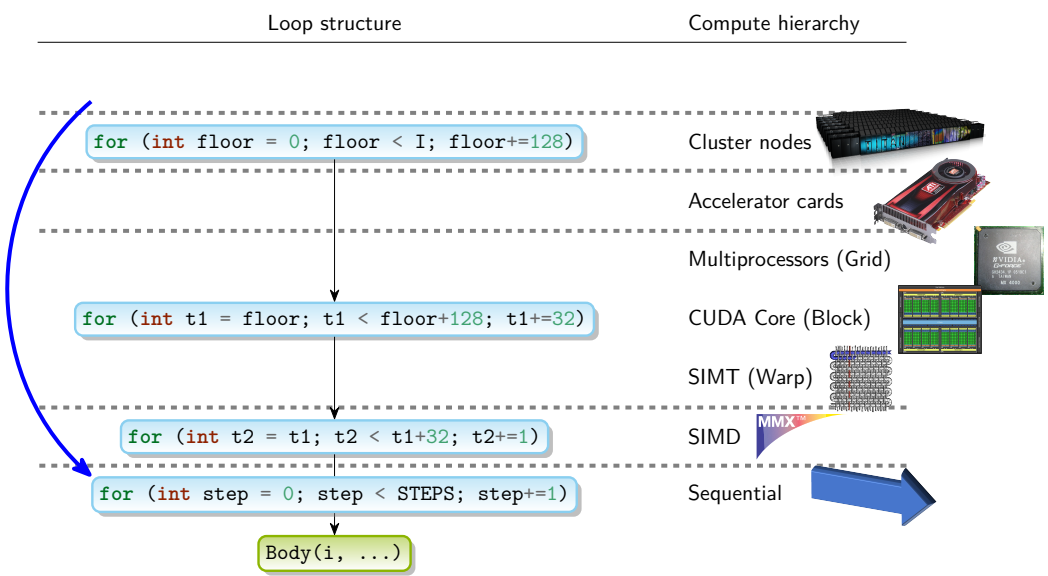
Compute Hierarchy Mapping



Compute Hierarchy Mapping



Compute Hierarchy Mapping



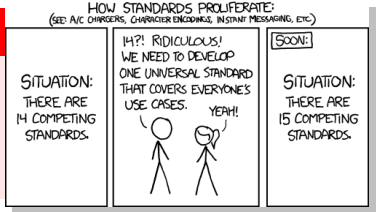
(Dis-)advantages

Advantages

- Speculatively apply transformations
  - Generalized legality check
  - Generalized profitability check
- Loop canonicalization only applied to DAG
- Assumptions not part of core structure
  - No per-transformation code versioning
- Independent of underlying IR

Disadvantages

- Cannot directly reference siblings/parents
  - Def/Use chains
- LLVM-IR, MachineIR, VPlan, MLIR, ...




That's all Folks!