Unified Sequential Optimization Directives in OpenMP

Brandon Neth (University of Arizona),
Thomas R.W. Scogland (Lawrence Livermore National Laboratory),
Michelle Mills Strout (University of Arizona), and
Bronis R. de Supinski (Lawrence Livermore National Laboratory)

My code is so slow!

Profiler?

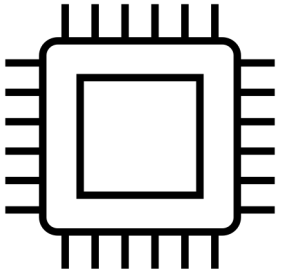
Hm, I'll inline

This program spends lots of time making function calls to `foo`.

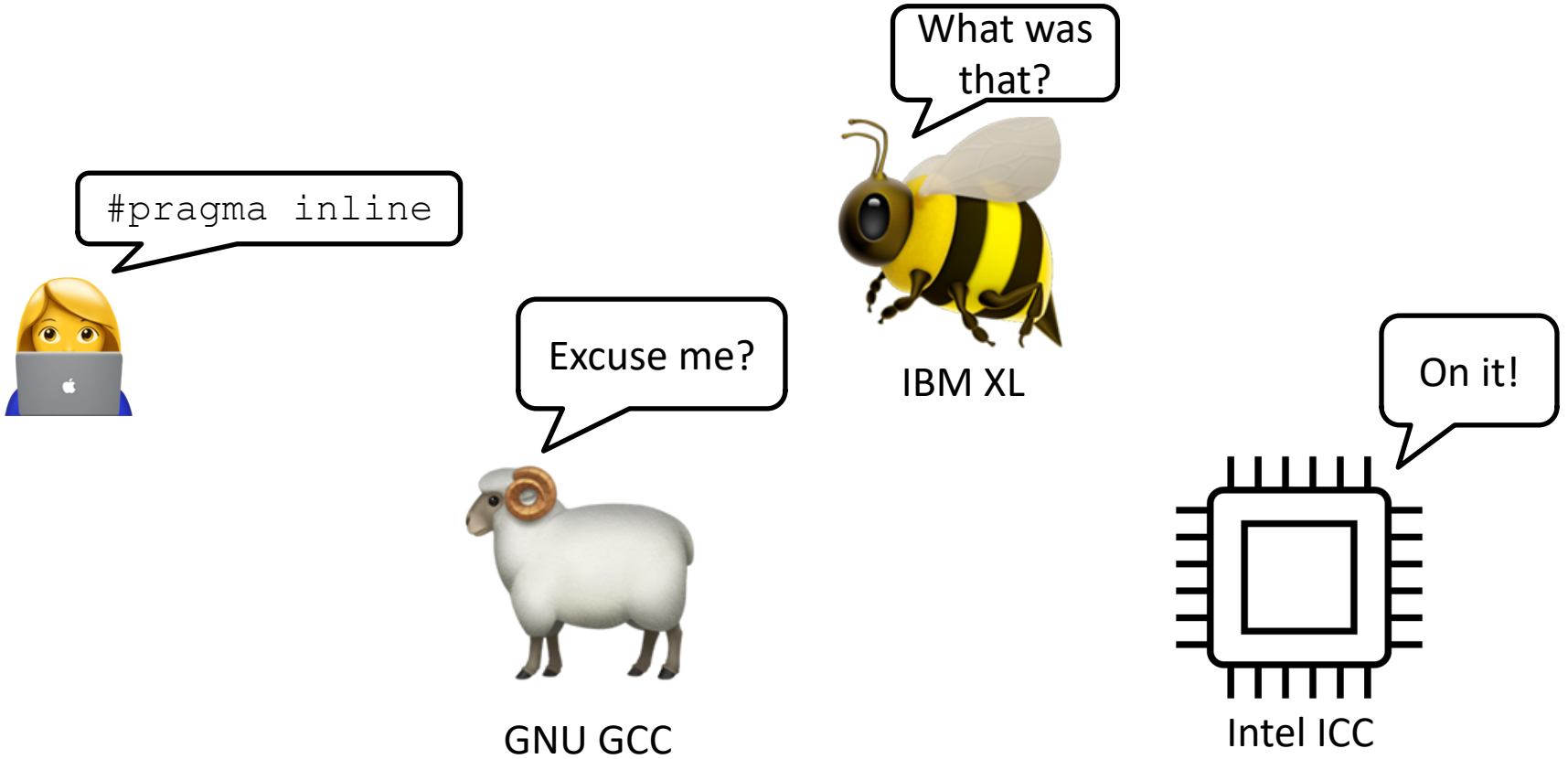
```
int important_and_complex_function(...)
{
  ...
  //deep in a parallel loop...
    foo(...);
  ...
}

double
popular_func_for_perf_critical_apps(...)
{
  ...
  important_and_complex_function(...);
  ...
}
...
```

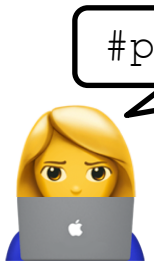
IBM XL

GNU GCC



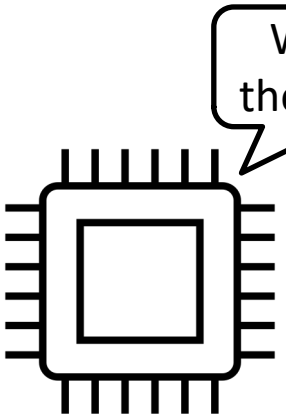
Intel ICC


#pragma inline
Excuse me?
GNU GCC
What was that?
IBM XL
On it!
Intel ICC

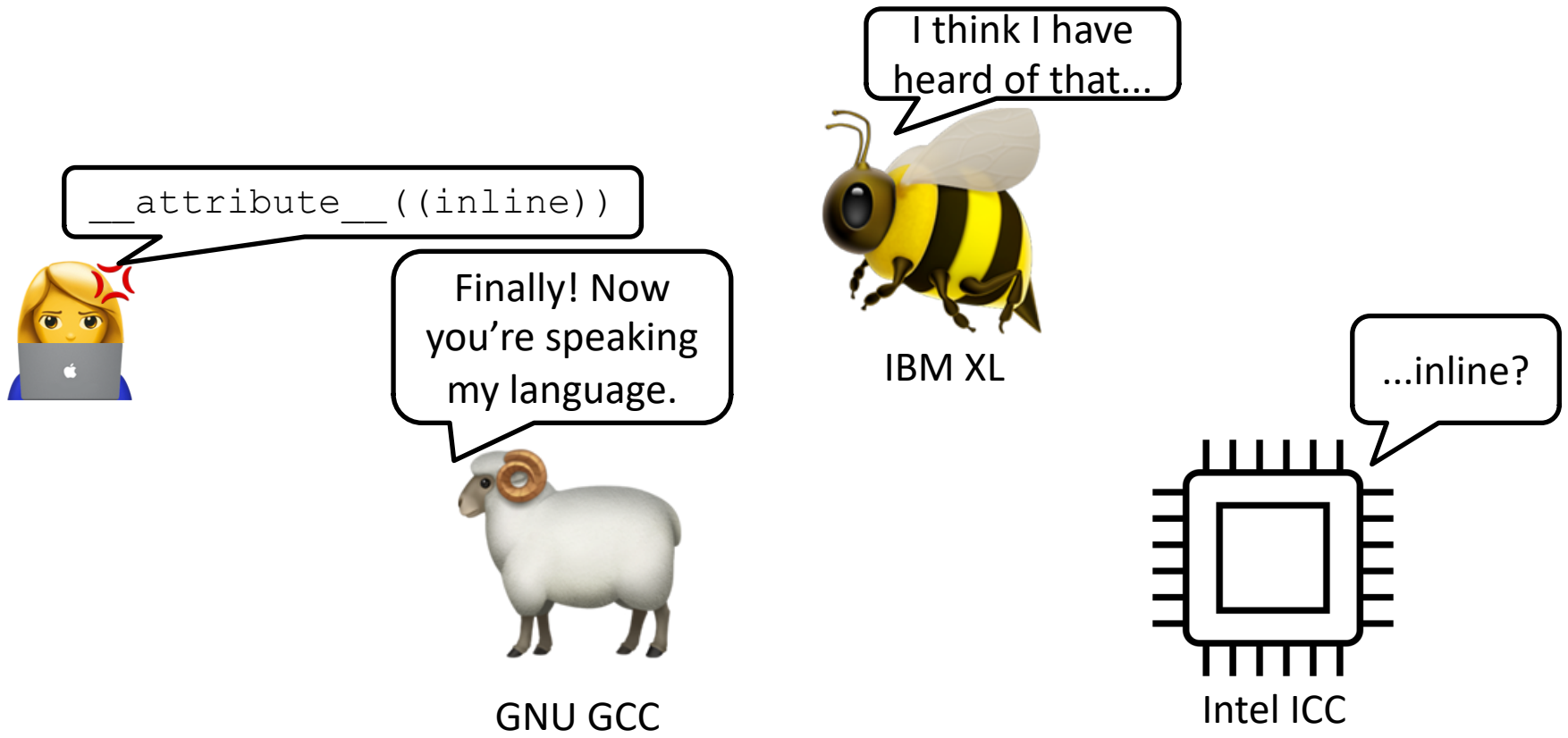


GNU GCC

IBM XL

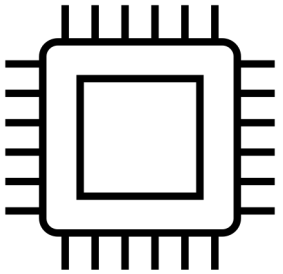


Intel ICC


__attribute__((inline))
Finally! Now you're speaking my language.
GNU GCC
I think I have heard of that...
IBM XL
...inline?
Intel ICC

IBM XL

GNU GCC



Intel ICC

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP’s history is one of standardization

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Inlining Directives

- Intel
  - `#pragma inline`
    - Where: At function call site
    - What: Encourages inlining
  - `#pragma forceinline recursive`
    - Where: At function call site
    - What: Forces inlining, applies recursively
- XL
  - `#pragma inline(foo)`
    - Where: file scope
    - What: Encourages inlining of `foo`
- GCC
  - `__attribute__((inline))`
    - Where: function declaration
    - What: Encourages inlining of attributed function
  - `__attribute__((always_inline))`
    - Where: function declaration
    - What: Forces inlining of attributed function
  - `__attribute__((flatten))`
    - Where: Function declaration
    - What: Inlines function calls within attributed function

Aliasing Directives

- Intel
  - None
- XL
  - `#pragma disjoint(a,b)`
    - Where: After declaration of identifiers
    - What: Listed identifiers do not share physical storage
- GCC
  - `__attribute__((alias ("target")))`
    - Where: Variable declaration
    - What: Indicates attributed variable aliases `target`

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Table 1. Average slowdowns when inlining is removed from RAJA Performance Suite, GCC.

Benchmark Category	Benchmark Count	Average Execution Time with Inlining (s)	Average Execution Time without Inlining (s)	Average Slowdown without Inlining
basic	10	0.45	1.37	3.03x
lcals	11	0.76	1.22	1.59x
polybench	13	0.88	20.46	23.23x
stream	5	1.31	1.62	1.23x
apps	7	1.05	3.12	2.97x
total	46	.79	3.30	4.18x

Table 2. Average slowdowns when inlining is removed from RAJA Performance Suite, Intel.

Benchmark Category	Benchmark Count	Average Execution Time with Inlining (s)	Average Execution Time without Inlining (s)	Average Slowdown without Inlining
basic	10	0.48	1.37	2.85x
lcals	11	0.93	1.64	1.75x
polybench	13	0.96	11.37	11.76
stream	5	1.04	1.05	1.01x
apps	7	0.83	3.37	4.04x
total	46	0.81	3.24	3.98x

Table 3. Execution Times and Binary Sizes for LULESH Variants

Compiler	Version	Average Execution Time (s)	Binary Size (kb)
GCC	Inlining	112.33	530
	No Directives	117.06	187
	No Inlining	115.14	315
Intel	Inlining	103.39	1490
	No Directives	108.87	732
	No Inlining	109.83	675

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Example 1: OpenMP Outlining

```
__attribute__((flatten))
void foo(int N) {
#pragma omp parallel for
  for(int i = 0; i < N; i++) {
    bar();
    baz();
  }
}
```

Example 2: Data sharing

```
void baz() {
  int a,b,c;
#pragma omp parallel
private(b)
  {
    foo(a,b,c);
  }
}

void foo(int a, int b, int
c)
{
#pragma omp task
  {
    ...
  }
}
```

```
void baz() {
  int a,b,c;
#pragma omp parallel
private(b)
  {
#pragma omp task
    {
      ...
    }
  }
}
```

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP's history is one of standardization

Current OpenMP features not related to shared memory loop-level parallelism

- V3.0 Tasking Constructs
- V4.0 SIMD Directives
- v5.0 Device Directives
- v5.1 Tile Construct

Concrete Syntax

- Aliasing
  - `#pragma omp aliases( list )`
  - `#pragma omp disjoint( list )`
- Inlining
  - `#pragma omp inline [ recursive ]`
  - `#pragma omp noinline`
- Side Effects
  - `#pragma omp pure`
  - `#pragma omp const`
- Alignment
  - `#pragma omp aligned [( alignment )]`
  - `#pragma omp begin aligned [( alignment )]`
  - `#pragma omp end aligned`

Sequential optimization directives belong in OpenMP

- Sequential optimizations directives need to be unified
  - Sequential optimization directives are syntactically and semantically fractured
  - Sequential optimizations are performance critical
- The unification should happen in OpenMP
  - Sequential optimizations interact with OpenMP constructs
  - OpenMP’s history is one of standardization