OpenMP Implementation in GCC

Diego Novillo

dnovillo@redhat.com

Red Hat Canada

GCC Developers Summit

Ottawa, Canada, June 2006

OpenMP

- Language extensions for shared memory concurrency (C, C++ and Fortran)
- Designed around compiler pragmas
  - **Directives** specify parallelism and work sharing
  - **Clauses** specify attributes for data sharing and scheduling
- Based on fork/join model

Programming Model

- Directives → **`#pragma omp`** (C, C++) or **`!$omp`** (Fortran)
- Compiler replaces directives with calls to runtime library (**`libgomp`**)
- Library offers API for querying/controlling threads and scheduling
- Runtime controls in program or environment variables
  - **`OMP_NUM_THREADS`**, **`OMP_SCHEDULE`**, **`OMP_DYNAMIC`**, **`OMP_NESTED`**

Programming Model

- Programmer responsible for synchronization and sharing
  - Sharing is variable based
  - Directives used for synchronization
- Original intent: Sequential run = parallel run
  - Compiler switch enables/disables the pragmas
  - Invalid sequential programs are possible too: parallel algorithms

OpenMP - Hello World

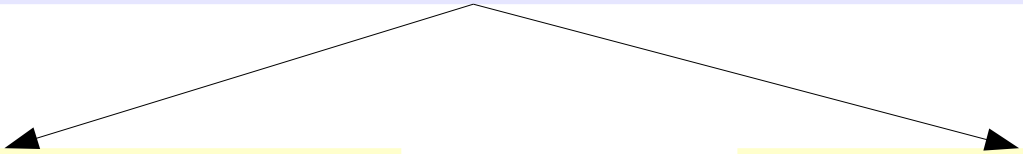
```
#include <omp.h>

main()

{

  #pragma omp parallel

  printf ("[%d] Hello\n", omp_get_thread_num());
}
```



```
$ gcc -fopenmp -o hello hello.c
$ ./hello
[2] Hello
[3] Hello
[0] Hello
[1] Hello
```

← Master thread

```
$ gcc -o hello hello.c
$ ./hello
[0] Hello
```

Distributing work across threads

- `#pragma omp for`

  data/loop parallelism

  Partitions iteration space with `schedule` clause

- `#pragma omp sections`

  `cobegin/coend` style parallelism

  Sections delimited with `#pragma omp section`

- `#pragma omp workshare`

  Distributes execution of Fortran FORALL, WHERE and array assignments

Data Sharing

- Various rules to determine sharing properties.
- Most are straightforward and intuitive.
  - Globals and heap allocated variables are shared
  - Locals declared inside a directive body are private
  - Loop iteration variables for parallel loops are private

Data Sharing

- Data sharing directive
  - `threadprivate`
- Data sharing clauses
  - `shared`
  - `private`
  - `firstprivate`
  - `lastprivate`
  - `reduction`

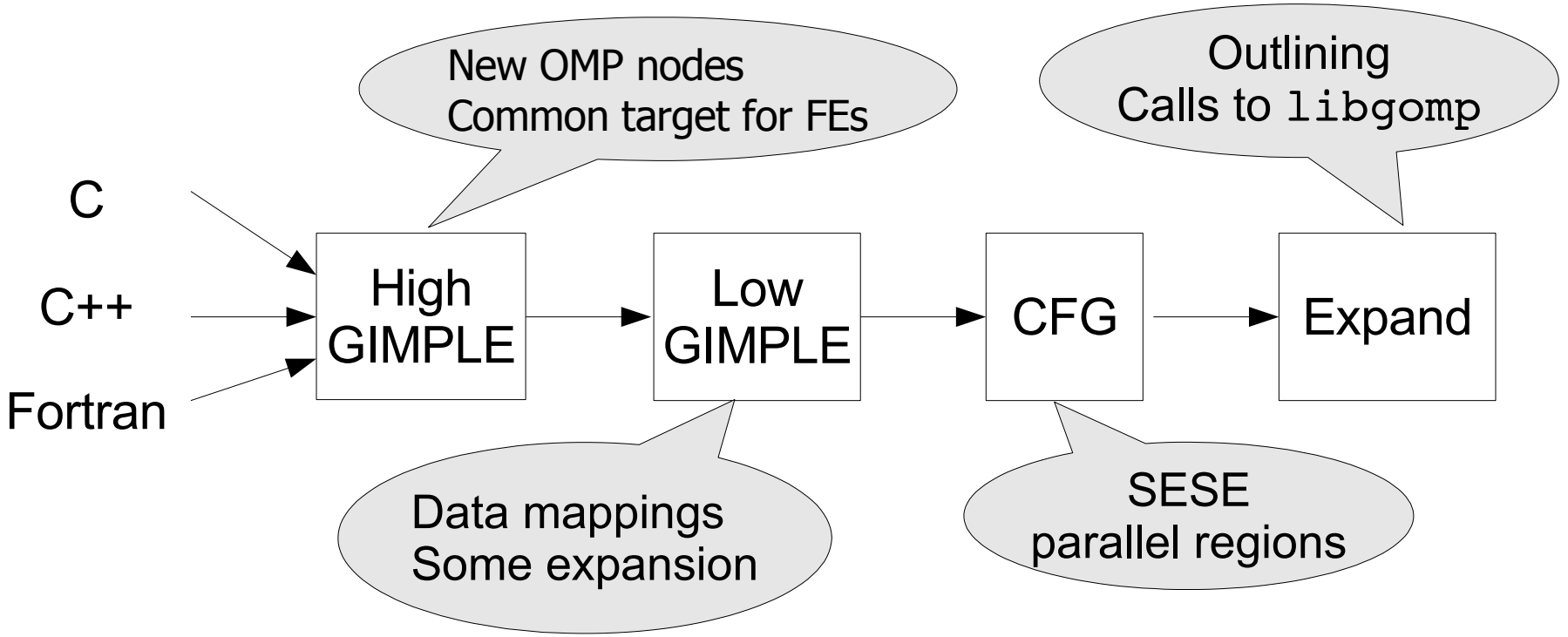
Synchronization

- `#pragma omp single`
- `#pragma omp master`
- `#pragma omp critical`
- `#pragma omp barrier`
- `#pragma omp atomic`
  - Atomic storage update: `x` *`op`*`= expr`, `x++`, `x--`
- `#pragma omp ordered`
  - Threads enter in loop iteration order.

Implementation

- GNU OpenMP (GOMP)
- Four components
  - Parsing
  - Intermediate Representation
  - Code Generation
  - Run time library (`libgomp`)

Implementation



Original Program

```
main()
{
  int i, sum = 0;

  #pragma omp parallel for
  for (i = 0; i < 10; i++)
   {
      #pragma omp atomic
      sum += i;
   }
  printf ("sum = %d\n", sum);
}
```

High GIMPLE

```
main ()
{
  int i, sum = 0;

  #pragma omp parallel shared(sum)
    {
      #pragma omp for nowait private(i)
      for (i = 0; i <= 9; i = i + 1)
        __sync_fetch_and_add_4 (&sum, i);
    }
  printf ("sum = %d\n", sum);
}
```

Low GIMPLE

```
main ()
{
  int i, *D.1576, sum = 0;
  struct .omp_data_s .omp_data_o;

  .omp_data_o.sum = &sum;
  #pragma omp parallel shared(sum)
  .omp_data_i = &.omp_data_o;
  #pragma omp for nowait private(i)
  for (i = 0; i <= 9; i = i + 1)
  D.1576 = .omp_data_i->sum;
  __sync_fetch_and_add_4 (D.1576, i.0);
  OMP_RETURN
  OMP_RETURN
  printf (&"sum = %d\n"[0], sum.1);
}
```

To be Outlined

Final expansion (`main`)

```
main ()
{
  int i, sum = 0;
  struct .omp_data_s .omp_data_o;

  .omp_data_o.sum = &sum;
  __builtin_GOMP_parallel_start (main.omp_fn.0,
                                 &.omp_data_o, 0);
  main.omp_fn.0 (&.omp_data_o);
  __builtin_GOMP_parallel_end ();
  printf ("sum = %d\n", sum);
}
```

Final Expansion (main.omp_fn.0)

```
main.omp_fn.0 (.omp_data_i)
{
 D.1581 = __builtin_omp_get_num_threads();
 D.1582 = (unsigned int) D.1581;
 D.1583 = __builtin_omp_get_thread_num();
 D.1584 = (unsigned int) D.1583;
 D.1585 = 10 / D.1582;
 D.1586 = D.1585 * D.1582;
 D.1587 = D.1586 != 10;
 D.1588 = D.1585 + D.1587;
 D.1589 = D.1588 * D.1584;
 D.1590 = D.1589 + D.1588;
 D.1591 = MIN_EXPR <D.1590, 10>;
 if (D.1589>=D.1591) goto <L2> else
   goto <L0>
<L2>:
 return;
```

Iteration space partitioning

```
<L0>:
  D.1592 = (int) D.1589;
  D.1593 = D.1592 * 1;
  i = D.1593 + 0;
  D.1594 = (int) D.1591;
  D.1595 = D.1594 * 1;
  D.1596 = D.1595 + 0;
<L1>:;
  D.1576 = .omp_data_i->sum;
  __sync_fetch_and_add_4 (D.1576, i);
  i = i + 1;
  D.1597 = i < D.1596;
  if (D.1597) goto <L1>; else goto <L2>;
}
```

Local min/max limits

Auto Parallelization

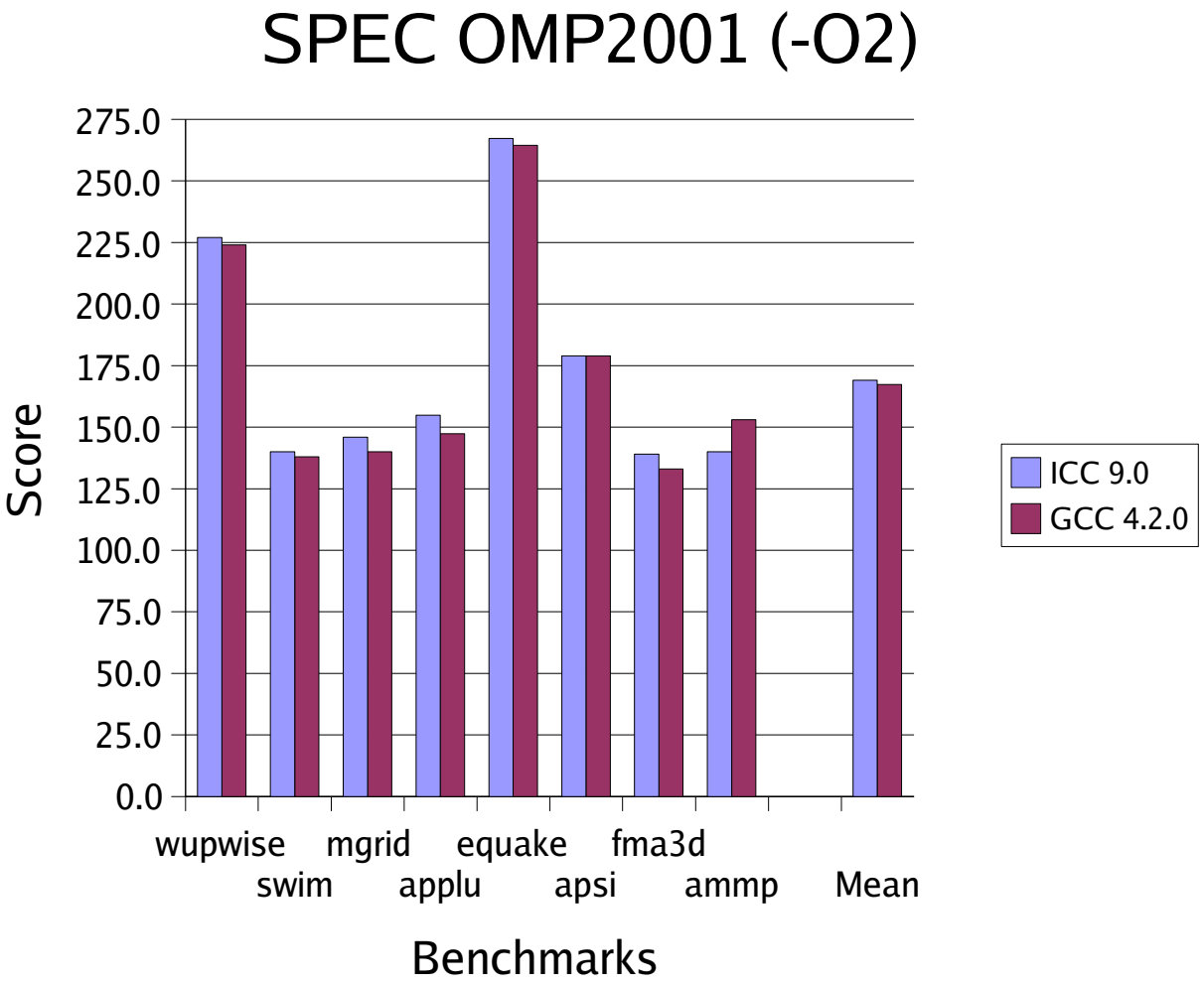
- OMP codes can be emitted before `pass_expand_omp`.
  - `OMP_SECTIONS` → task parallelism
  - `OMP_FOR` → loop parallelism
  - `OMP` sharing clauses → data sharing
  - Synchronization with appropriate directives
- Code should be in normal form
  - Works in low GIMPLE
  - CFG is available

Runtime library

- Wrapper around POSIX threads
  - Various system-specific performance tweaks
- Synchronization usually 1-1 mapping except
  - `omp master` → Blocks threads with ID != 0
  - `omp single` → `copyprivate` needs special expansion to handle broadcast.
- All scheduling variants of `omp for` implemented
  - Static schedules are open coded by compiler

Status and Future Work

- To be released with 4.2 later this year
- Implementation available in Fedora Core 5
- Automatic parallelism planned using OpenMP infrastructure
- Usual tweaks and bug fixing


SPEC OMP2001 (-O2)
Score
275.0
250.0
225.0
200.0
175.0
150.0
125.0
100.0
75.0
50.0
25.0
0.0
wupwise
swim
mgrid
applu
equake
apsi
fma3d
ammp
Mean
Benchmarks
ICC 9.0
GCC 4.2.0