Hipeac2 Programming Models and Operating Systems

Source-2-Source Program Transformations with the Mercurium Compiler

BSC-UPC

Hipeac2 cluster meetings,
Paris, Nov. 27th-28th, 2008

Outline

- Need for program transformations
- The Mercurium compiler
- Example: supporting Cell SPUs
- Coding applications for the Cell
- Summary

Need for program transformations

#pragma omp parallel

{

i

l

(

…

}

#pragma omp parallel firstprivate(niter) private(step)

{

}

```
for (Row = 0; Row < QuadrantSize; Row++) {
        for (Col = 0; Col < QuadrantSize; Col++) {
#pragma omp task untied if(Depth < nbs_cutoff_value) firstprivate(Row, Col, Depth)
                                                          private (Vertical, Horizontal)
            { float *ARowStart = A;
              float Sum0 = ...
              for (Vertical = 0; Vertical < MatrixSize; Vertical++) {
                  for (Horizontal = 0; Horizontal < MatrixSize; Horizontal += 8) {
                      float *BColumnStart = B + Horizontal;
                      for (Products = 0; Products < MatrixSize; Products++) {
                          float ARowValue = *ARowStart++;

                          Sum0 += ARowValue * (*BColumnStart);
                          Sum1 += ARowValue * (*(BColumnStart+1));
```

Need for program transformations

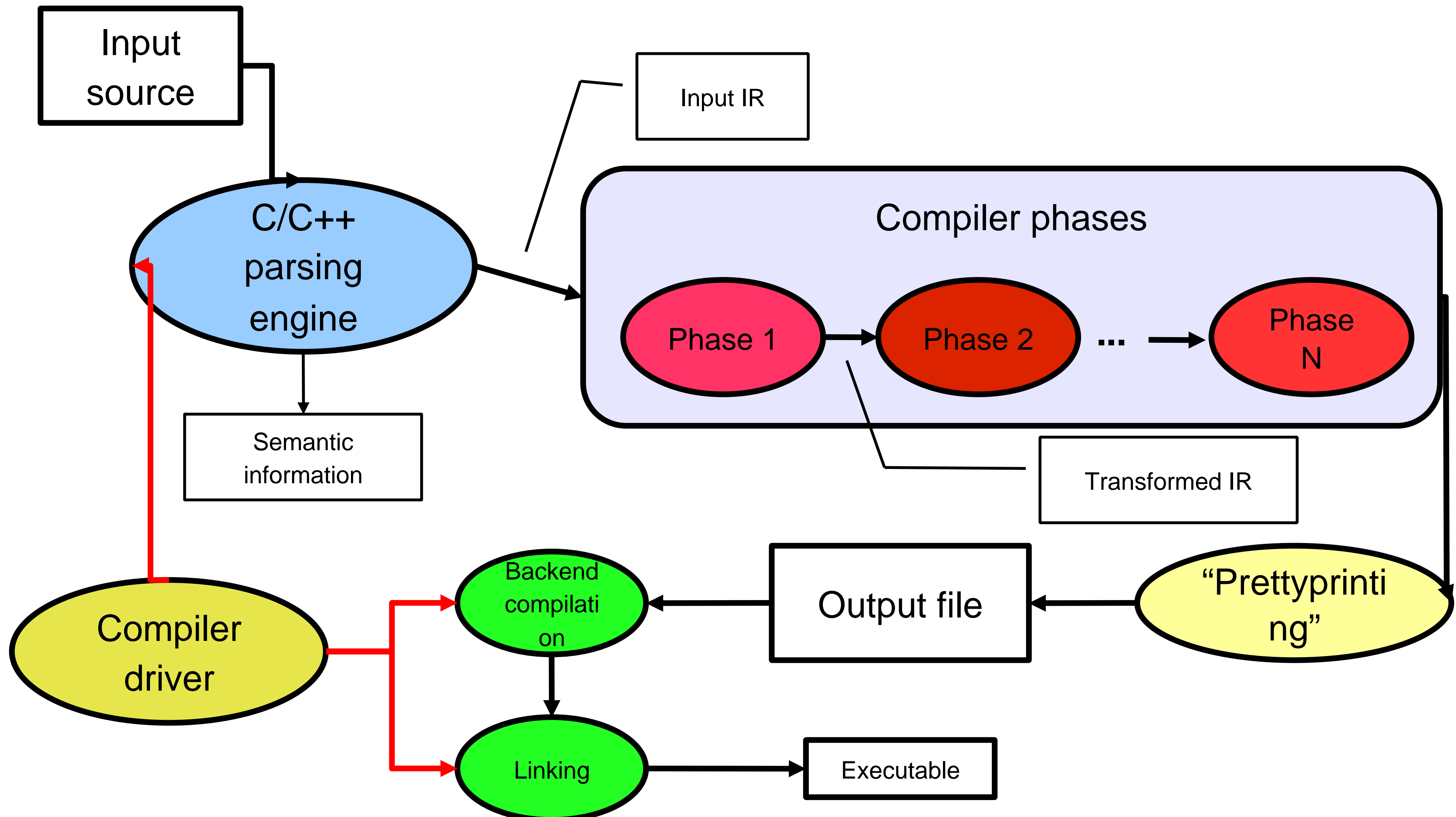
```
#pragma omp parallel
{
  ini
  lhs
  ex
...
}
```

```
nth_team_t nth_current_team;
int nth_task_type = NTH_DTYPE_LOCAL;
/* Creating team */
nth_init_team(&nth_current_team, nth_team_size, nth_selfv, nth_team_data);
nth_num_deps = 0;
int nth_p;
for (nth_p = 1; nth_p < nth_team_size; nth_p++) {
    /* Master creates team members */
    nth_desc_t * nth = nth_create((void *) (nth__main_1), &nth_task_type,
                                  &nth_num_deps, &nth_p, &nth_selfv, &nth_arg_addr_ptr,
                                  &nth_nargs_ref, &nth_nargs_val, &nth_current_team);
    nth_submit(nth);
}
    /* Master invokes outline */
    (nth__main_1)(&nth_current_team);
/* Ending team */
nth_end_team(&nth_current_team);
```

Need for program transformations

```
for (Row = 0; Row < QuadrantSize; Row++) {
    for (Col = 0; Col < QuadrantSize; Col++) {
            /* Create the task */
            nth_desc * nth;
            int nth_type = NTH_DTYPE_TEAM;
            int nth_ndeps = 0;
            int nth_vp = 0;
            nth_desc_t * nth_succ = (nth_desc_t *) 0;
            int nth_nargs_ref = 0;
            int nth_nargs_val = 5;
            void * nth_arg_addr[5 + 1];
            void * * nth_arg_addr_ptr = &nth_arg_addr[1];
            size_t nth_size[] = {0, sizeof (M5), sizeof (S1), sizeof (S5),
                                        sizeof (QuadrantSize), sizeof (Depth)};
            nth = nth_create_task((void *) (nth__OptimizedStrassenMultiply_par_2), &nth_type,
                    &nth_ndeps, &nth_vp, &nth_succ, &nth_arg_addr_ptr, &nth_nargs_ref,
                    &nth_nargs_val, &nth_size[1], &M5, &nth_size[2], &S1, &nth_size[3],
                    &S5, &nth_size[4], &QuadrantSize, &nth_size[5], &Depth);
            nth_submit (nth);
```

Compiler flow



SCALABLE COMPUTER ARCHITECTURE
SARC

Characteristics

- Developed in C++
- Parsing C and C++ supported
  - Most of gcc extensions
  - Generates IR (AST, symbolic and scoping info)
- Transformation passes
  - Change IR
  - Can add new IR for later phases
- Source-2-source: Pretty-printing AST

SCALABLE COMPUTER ARCHITECTURE SARC

Phases support

- Easy to add new phases to the compiler
  - Can inherit from CompilerPhase class
  - Receive current IR
  - Return new IR
- Configuration file indicates which phases must be executed in a per-compilation basis

SARC
SCALABLE COMPUTER ARCHITECTURE

Phases support

- Common transformations abstracted away
  - OpenMPPhase, to extend OpenMP
  - PragmaCustomCompilerPhase, to implement your own set of directives
- Common language constructs wrapped in classes for easier manipulation
  - FuncDef, Decl, Expr, Statement...

Example: supporting Cell SPUs

- Target architecture is h... rogeneous
  - Need multifile supp...

```
void spu_outlined_default3 (double a[Nelems+0], double c[Nelems+0], int j)
{
       {
           spu_get_dma_array_section_1_tag(_lp_a, a, sizeof(double), j, j + ((blksize) - 1) * (1), 13);
           spu_dma_wait_tag(13);
       }
       {
           int _blk_j;
           for (_blk_j = j;
               _blk_j <= ((((Nelems) - 1) < (j + ((blksize) - 1) * 1)) ? ((Nelems) - 1) : (j + ((blksize) - 1) * 1));
               _blk_j += (1))
           {
               (*_lp_c)[(_blk_j) - (j)] = (*_lp_a)[(_blk_j) - (j)];
           }
       }
      ...
}
```

SARC
SCALABLE COMPUTER ARCHITECTURE

Example: supporting Cell SPUs

- Prettyprinting part of the AST onto different files
- New files generated posted for later processing
- Two different profiles in config file for Cell/B.E. invoking different backend compilers (gcc, xlc)
  - cellmpcc -> ppu-gcc, ppuxlc
  - spucellcc -> spu-gcc, spu-xlc

Example: supporting Cell SPUs

- Spawning tasks to SPUs

```
for (j=0; j < N; j++)
{
  #pragma spu task
  {
    #pragma spu data input(a[j], b[j])
    #pragma spu block nest(1) factors(BK)
    {
      c[j] = a[j] + b[j];
    }
    #pragma spu data output(c[j])
  }
}
```

SARC
SCALABLE COMPUTER ARCHITECTURE

Example: supporting Cell SPUs

- Main CPU transformation layout

```
Source spawn_code, data_environment, spawn_call;
spawn_code
     << "{"
     <<    data_environment
     <<    spawn_call
     << "}"
     ;

spawn_call
     << "int function = &" << function_name
     <<                                "_" << function_id << ";"
     << "css_addTask(function, " << css_parameter_list << ");"
     ;
```

int function = &__worker_function_id_3;
css_addTask(function, a, b, c, j);

SCALABLE COMPUTER ARCHITECTURE SARC

Example: supporting Cell SPUs

- Outlined function spawned to target SPU

```
outlined_function
     << headers
     << "void " << function_name << "(" << data_parameters << ")"
     << "{"
     <<       outlined_body_source
     << "}"
     ;
```

```
void spu_outlined_default3 (double a[Nelems+0], double b[Nelems+0], double c[Nelems+0], int j)
{   ...
      spu_get_dma_array_section_1_tag(_lp_a, a, sizeof(double), j, j + ((blksize) - 1) * (1), 13);
      spu_get_dma_array_section_1_tag(_lp_b, b, sizeof(double), j, j + ((blksize) - 1) * (1), 13);
      spu_dma_wait_tag(13);

      int _blk_j;
      for (_blk_j = j;
          _blk_j <= ((((Nelems) - 1) < (j + ((blksize) - 1) * 1)) ? ((Nelems) - 1) : (j + ((blksize) - 1) * 1));
          _blk_j += (1))
       {
            (*_lp_c)[(_blk_j) - (j)] = (*_lp_a)[(_blk_j) - (j)] + (*_lp_b)[(_blk_j) - (j)];
       }
   ...
}
```

Example: supporting Cell SPUs

- Collecting parameters of the spawned function

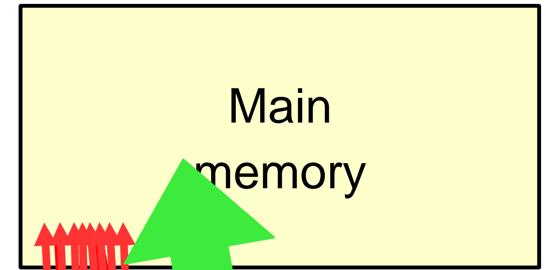
```
ObjectList<IdExpression> id_expressions = input_clause.id_expressions();
for (ObjectList<IdExpression>::iterator it = id_expressions.begin();
        it != id_expressions.end();
        it++)
{
    Symbol sym = it->get_symbol();

    Type type = sym.get_type();

    Source parameter_decl;
    parameter_decl << type.get_declaration(
                it->get_scope(),
                it->prettyprint(),
                Type::PARAMETER_DECLARATION);

    data_parameters.append_with_separator(parameter_decl, ",");
}
```

1) For every variable in the 'input' clause

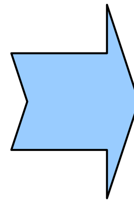
2) Get the type of this variable

3) Use it to get a parameter declaration of the same name

Coding STREAM




```
void tuned_STREAM_Triad(double scalar)
{
       int j;
#pragma omp parallel for
       for (j=0; j<N; j++)
              a[j] = b[j] + scalar*c[j];
}
```



```
void tuned_STREAM_Triad(double scalar)
{
       int j;
#pragma omp parall
       for (j=0; j<N; j++)
#pragma spu task
         {
#pragma spu data input (b[j], c[j])
#pragma spu block nest(1) factors (4885)
            {
              a[j] = b[j] + scalar*c[j];
            }
#pragma spu data output (a[j])
         }
       }
}
```

SCALABLE COMPUTER ARCHITECTURE SARC

Coding applications for the Cell

- Section of code from PBPI

```
for (iPattern = 0; iPattern < nPatt; iPattern += 4)
{
#pragma spu task
  {
#pragma spu data input(tl[0:15],tr[0:15],tp[0:15],  pl[iPattern:iPattern+3], \
                         pr[iPattern:iPattern+3],    pp[iPattern:iPattern+3])
#pragma spu block nest(1) factors(blksize)
    {
      t1 = pl[iPattern+0] * tl[0] + pl[iPattern+1] * tl[1] +
           pl[iPattern+2] * tl[2] + pl[iPattern+3] * tl[3];
      ...
      t3 = pp[iPattern+0] * tp[12] + pp[iPattern+1] * tp[13] +
           pp[iPattern+2] * tp[14] + pp[iPattern+3] * tp[15];
      pn[iPattern+3] = t1 * t2 * t3;
    } // block
#pragma spu data output (pn[iPattern:iPattern+3])
  } //task
} //for
```

SCALABLE COMPUTER ARCHITECTURE
SARC

Current status

- OpenMP, most of 2.5, 3.0 tasking
- CellMP, input/output/blocking Cell extensions
- Acotes streaming transformations
- Cell/SMP Superscalar
- Prototype for FPGAs
- Prototype for GPUs

SCALABLE COMPUTER ARCHITECTURE SARC

Summary

- Mercurium: s-2-s compilation system
- Supporting a variety of transformations
- Flexible to add new transformation phases
- Easies experimentation within current architectures

SCALABLE COMPUTER ARCHITECTURE SARC

Q&A

- Thanks for your attention

- Contact us: mercurium@ac.upc.edu
- Compiler link: http://www.bsc.es