smecc Compiler Overview

SMECY Artemis European project

Ronan KERYELL (Ronan.Keryell@silkan.com)
Vincent LANORE (vincent.lanore@ens-lyon.fr)

SILKAN Wild Systems
4962 El Camino Real #201
Los Altos, CA 94022, USA

2013/04/24

SILKAN

- HPC Project: startup created in 2007 by people from HPC and compilation
- Recent fusion with Arion (real-time bus technology) and IlGeometry (polynomial-based geometrical modeling)
  ⇝ SILKAN
- Company specialized in high-end scientific simulation
  - Numerical models
  - High-performance libraries (sparse linear algebra...)
  - Virtual reality with real-time physics simulation for training
  - Hardware-in-the-loop with avionics certification (landing gear workbench...)
  - Model compilation and parallelization of Scilab, Matlab, Python, C, Fortran
  - Professional services (developments, training)
- 70+ people, mainly in France (Paris, Montpellier...)
  - 15 in Montréal (Canada)
  - 5 in Los Altos (USA)

SILKAN

Context

- Previous development of massively parallel infrastructures for HPC
- Power "Wall" ⇝ no longer possible to power fast sequential processors
- Now: massively parallel system on chip in embedded systems
  - ST+CEA STHORM: 2 ARM Cortex-A9 + fabric with 16 clusters of 16+1 PE (with SILKAN in SMECY project)
  - Kalray MPPA: 288 PE (with SILKAN in SIMILAN project)
  - Tilera TILEPro, TILE64, TILE-Gx: ≈64 PE (QOSMOS, OpenMP)
  - Adapteva Epiphany
  - ...

  (there was interesting presentations on these at "GdR SoC-SiP on manycores' day", last week)
- IP providers need to re-target existing solutions
- Need new software + hardware components

⇝ Motivation for SMECY Artemis European project

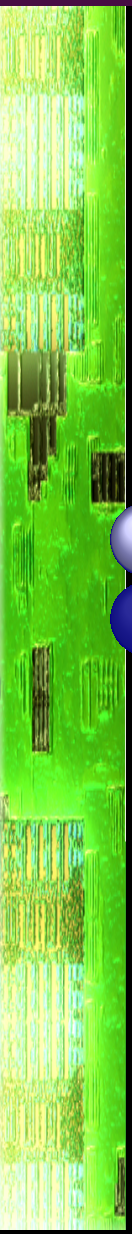
SILKAN

SMECY Artemis European project

- 30+ partners
- 2010/2/1–2013/2/1 (36 months)
- Total costs: 20.5 M€
- SILKAN mainly interacts with
  - Thales TRT: SPEAR-DE data-flow software environment & definition of SME-C
  - ST: STHORM platform + SDK with OpenCL
  - CEA: definition of SME-C & MCAPI implementation
  - ACE: definition of SME-C
  - SELEX SI: radar application provider
  - UTIA: ASVP platform (FPGA-based vector DSP)
  - BUT: RAVAC vectorizer compiler based on LLVM

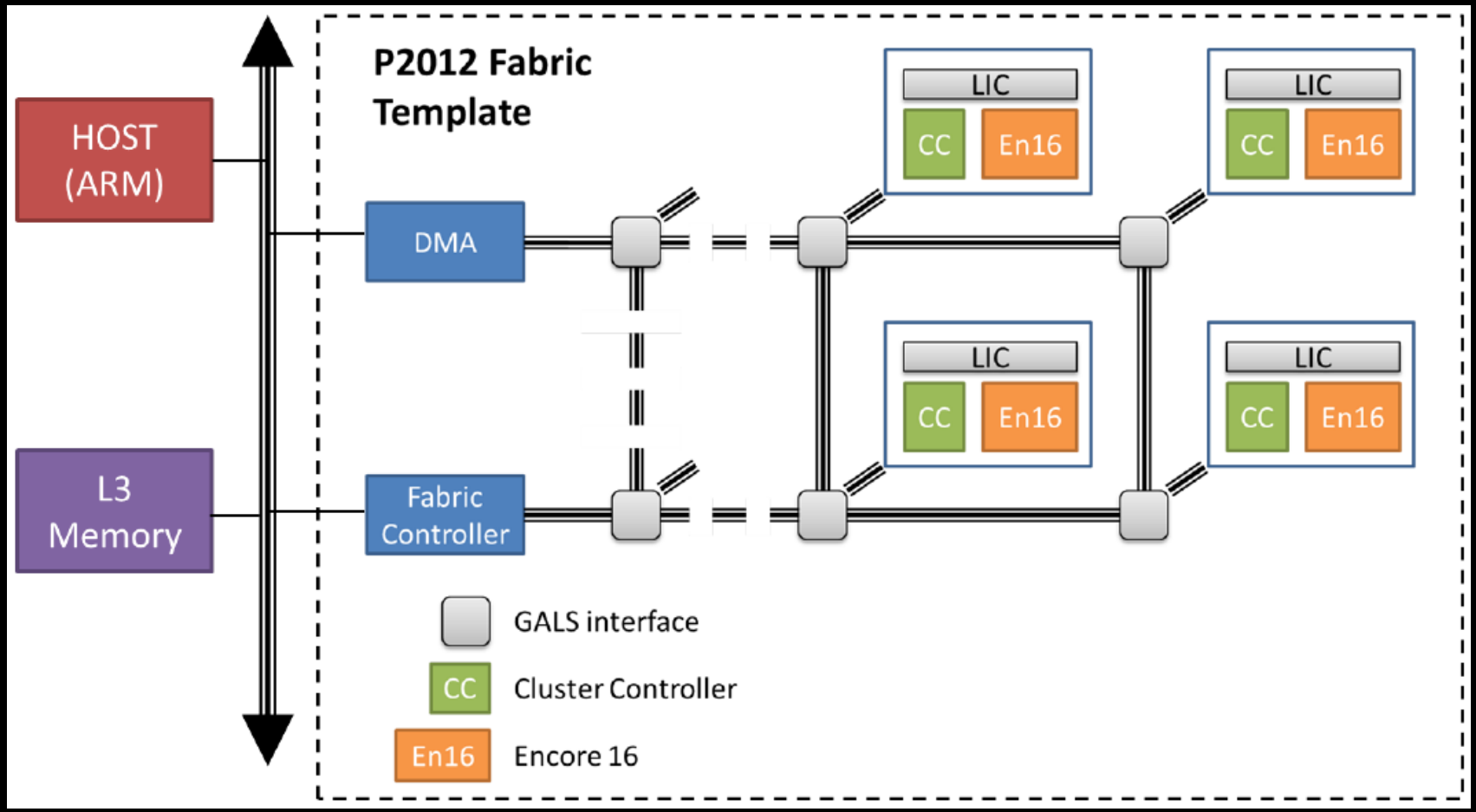
SILKAN

Outline

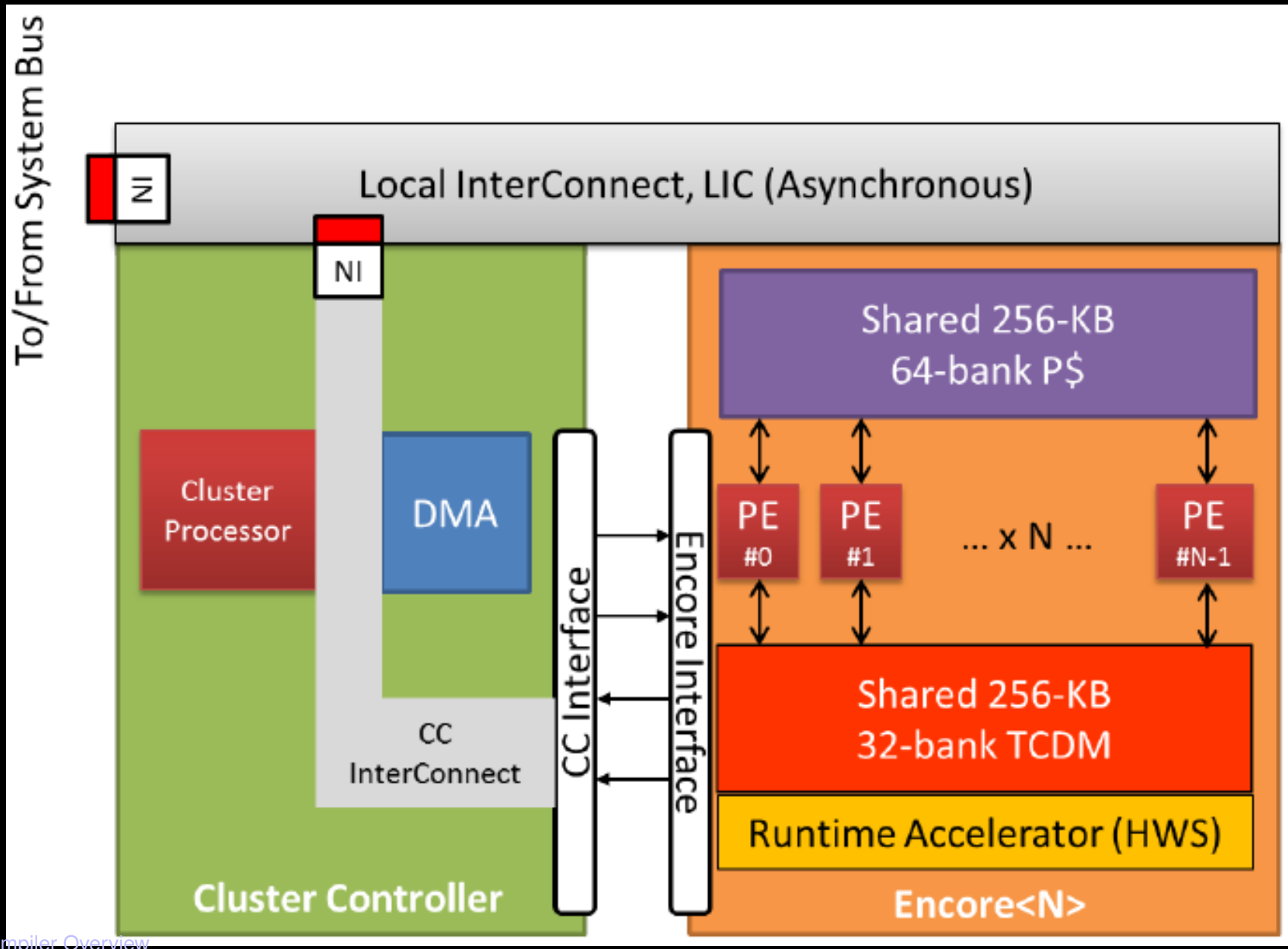


SILKAN

ST STHORM (P2012)



STHORM cluster architecture



Outline

SILKAN

OpenMP (I)

- “Multithread for dummies” ☺
- Target SMP (shared memory processors)
- Basic parallel SPMD execution based on *fork*/*join*
- No need to use system threads
- Idea: powder a program with directives to help the compiler to parallelize
- Philosophy: `#pragma`tism with aestheticism
-  If no directive, no parallelism (*a priori*)
-  Directive ≡ sworn statement
- Supported languages: Fortran & C/C++

SILKAN

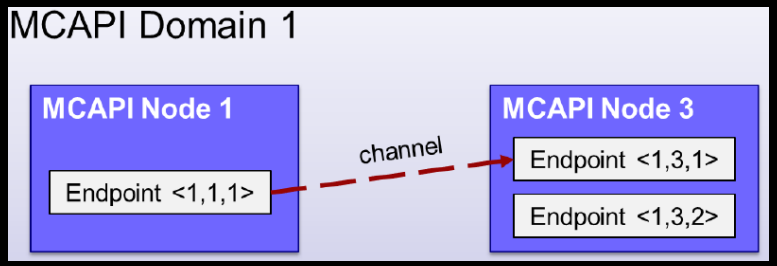
OpenMP (II)

```
1   /* This is executed by several threads in // */
    #pragma omp parallel for
3     for (i=0; i<n; i++)
          // Iterations are distributed between the threads
5         x[i] += y[i];
    // Implicit synchronization here
7
  #pragma omp parallel
9 {
    #pragma omp single private(p)
11  {
      p = listhead ;
13    while (p) {
        #pragma omp task
15      {
          process(p);
17      }
        p = next(p);
19    }
  }
21 }
```

- More heterogeneous computing support in OpenMP 4

MultiCore Association API target (I)

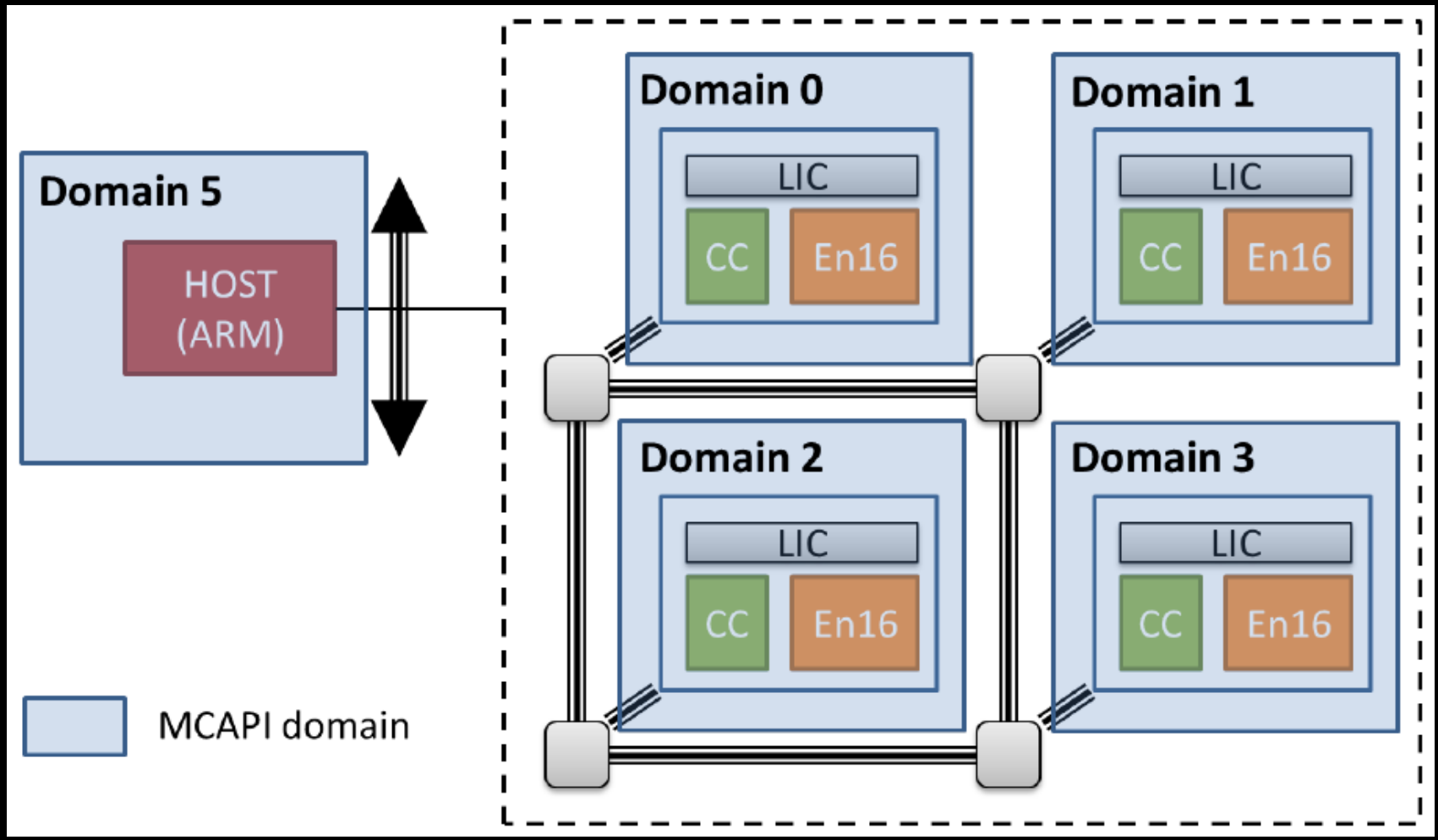
- Industrial standard
- Kind of poor-man MPI for embedded systems
- Natural target for STHORM: ST decided to switch from their own library to MCA API
- Message passing library connecting nodes
  - MCAPI: Communications
  - MTAPI: Threads, synchronization...
  - MRAPI: Resource management...



- Nodes can be software or hardware ⇝ Light weight
- Different implementations
  - Reference implementation for Linux done by a **woman** programmer formerly at Freescale
  - Implementation for STHORM by CEA. On-going...

SILKAN

MCAPI on STHORM

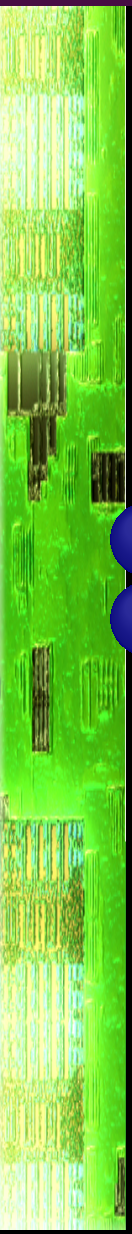


OpenCL

- Standard for GPU and many different accelerators
- Kernel-oriented computations on streams
- Data-parallelism and control-parallelism (1–3-dimensions) according to targets
- Complex split memory model, close to GPU...
- New types (vectors, images...)
- Need a lot of code restructuring
- ∃ C++ wrappers to simplify house-keeping code
- Portable programming standard...
- ... But different constraints on various targets
- ... Need different OpenCL writing style for performance according to the target
- ∃ OpenCL implementation for STHORM by ST

SILKAN

Outline

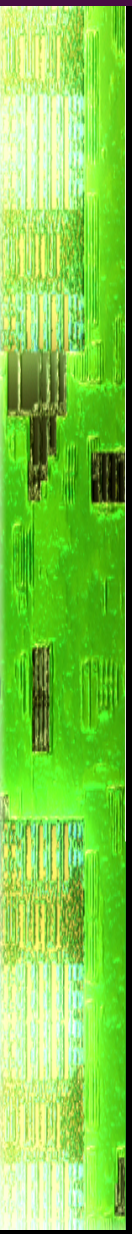


SILKAN

¿Internal representation?

- Many programming tools & languages (p)
- Many targets (t)
- Need $p \times t$ compilers?
- If central representation: only $p + t$ compilers
- If internal representation human-compatible: can even be used as a programming language!
- If `#pragma` based-language
  - Not yet-another-language to learn
    - $\rightsquigarrow$ Better acceptance
  - If the `#pragma` does not change the semantics: functional emulation for free! ☺
  - If OpenMP-based `#pragma`: parallel execution on SMP with a simple OpenMP compiler! ☺

SME-C pragma extensions



- Add few `#pragma smecy` to OpenMP 3.1 `#pragma`
- Add mapping information on where a function is to be run
- Add data usage information
- Add pipelining information for loops

SILKAN

SME-C mapping example (I)

```
1  #include <stdio.h>
2
3  #define N 10
4  #define M 5
5
6  void init(int* array, int size, int scale) {
7    for (int i = 0; i < size; i++)
8      array[i] = i*scale;
9  }
10
11 int main() {
12   int tab[N][M];
13   /* The schedule(static, 1) enforces each iteration to be executed in a
14      different thread, whatever the number of CPU is: */
15 #pragma omp parallel for schedule(static, 1)
16   for (int i = 0; i < N; i++) {
17     // Map on STHORM cluster 0 PE i:
18 #pragma smecy map(STHORM, 0, i)          \
19                arg(1,out,[N][M],/[i][]) \
20                arg(2,in) \
21                arg(3,in)
22     init(&tab[i][0], M, i+1);
23   }
```

SME-C mapping example (II)

```
25    for (int i = 0; i < N; i++) {
        printf("Line␣%d␣:", i);
27      for (int j = 0; j < M; j++)
          printf("%d␣", tab[i][j]);
29      puts("");
      }
31    return 0;
}
```

SME-C loop pipeline example (I)

```c
#include <stdio.h>

/* Generate counting numbers */
void generate(int * thing, int * product) {
  static int state = 0;
  *thing = state++;
  *product = 3;
  printf("\t*** Generate: %d (%p)\n", *thing, thing);
}


/* Some basic transformation */
void transform(int * thing, int * product) {
  *thing *= *product;
  printf("\tvvv Transform: %d (%p) with %d (%p)\n", *thing, thing, *product
}

/* Consume the flux */
void consume(int * thing) {
  printf("\t>>> Consume: %d (%p)\n", *thing, thing);
}


int main() {
```

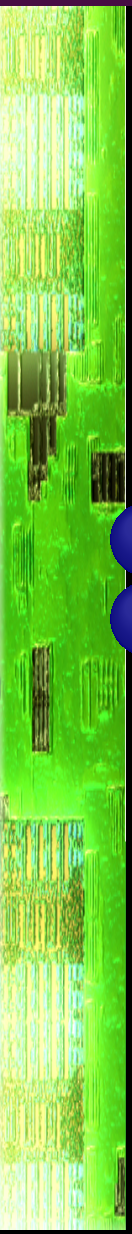
SME-C loop pipeline example (II)

```
  int b[1] = { sizeof(int) };
26 int c[1] = { 3 };

28 #pragma smecy stream_loop
  while (1) {
30 #pragma smecy stage arg(1,out) arg(2,out) map(PE,1)
    generate(b, c);
32 #pragma smecy stage label(2) arg(1,inout) arg(2,in)
    transform(b, c);
34 #pragma smecy stage arg(1,in)
    consume(b);
36 }

38 return 0;
}
```

Outline



SILKAN

smecc (I)

- Opportunity to investigate in Open Source source-to-source compiler ROSE
- Try C++11, STL, Boost...
- Parse the `#pragma`
- Generate an intermediate output: IR2, *à la* Par4All Accel
  - Ease re-targeting
  - Can be emulated in OpenMP !
  - Ease debugging
- Follow-up of PHRASE & CoMap project $\approx$ 2005 @ TÉLÉCOM Bretagne implemented in PIPS source-to-source framework from MINES ParisTech

SILKAN

Application code sample generated by SPEAR (I)

```
#pragma omp parallel for private (i_F_0,idxTime) \
        num_threads(4) schedule(static,1)
for (i_F_0 = 0; i_F_0 < 21;  i_F_0++) {
  threadprivate_i_F_0 = i_F_0;
  #pragma omp critical
  {
    // Pick a data block
    BlockCov(128, 8, 16,
             (Cplfloat(*)[128])&UG->SegHost.S1.Vect2Cov_out[0 + i_F_0*1][0]
             (Cplfloat(*)[16][8][8])&UG->SegHost.S1.S2.F_BlockCov_out[0][0]
    [...]
  }
  int i_F_R_0;
  idxTime = 0;
  for (i_F_R_0 = 0; i_F_R_0 < 16;  i_F_R_0++) {
    #pragma smecy map(STHORM,(threadprivate_i_F_0)%4,0) \
                  arg(2, [8][8], in) arg(3, [8][8], out)
    ComputeCholesky(8,
      (Cplfloat(*)[8])&UG->SegClusters[(i_F_0)%4].S3.F_X_2_out[0][0][0][0],
      (Cplfloat(*)[8])&UG->SegClusters[(i_F_0)%4].S3.S4.F_R_BlockChol_out[0
    #pragma smecy map(STHORM,(threadprivate_i_F_0)%4,0) \
                  arg(2, [8][8], in)  arg(3, [8][8], out)
    Invert(8,
      (Cplfloat(*)[8])&UG->SegClusters[(i_F_0)%4].S3.S4.F_R_BlockChol_out[0
```

SILKAN

Application code sample generated by SPEAR (II)

```
25          (Cplfloat(*)[8])&UG->SegClusters[(i_F_0)%4].S3.S4.F_R_BlockInvert_out
        [...]
```

SME-C IR2 output from smecc (host side) (I)

```
int main(int argc,char *argv[])
{
  SMECY_initialize_then_finalize();
  [...]
}
[...]

#pragma omp parallel for private ( i_F_0, idxTime ) num_threads ( 4 ) \
        schedule ( static, 1 )
for (i_F_0 = 0; i_F_0 < 21; i_F_0++) {
  threadprivate_i_F_0 = i_F_0;

  #pragma omp critical
  {
    // Pick a data block
    BlockCov(128,8,16,
             ((Cplfloat (*)[128UL])(UG -> SegHost.S1.Vect2Cov_out[0 + (i_F_
             ((Cplfloat (*)[16UL][8UL][8UL])(UG -> SegHost.S1.S2.F_BlockCov_
    [...]
  }
  int i_F_R_0;
  idxTime = 0;
  for (i_F_R_0 = 0; i_F_R_0 < 16; i_F_R_0++) {
    SMECY_set(ComputeCholesky,5,STHORM,(threadprivate_i_F_0 % 4),0);
```

SME-C IR2 output from smecc (host side) (//)

```
    SMECY_send_arg(ComputeCholesky,1,int,8,STHORM,(threadprivate_i_F_0 % 4)
26  SMECY_send_arg_vector(ComputeCholesky,2,Cplfloat,
      ((Cplfloat (*)[8UL])((UG -> SegClusters)[i_F_0 % 4].S3.F_X_2_out[0][0]
28    8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
    SMECY_prepare_get_arg_vector(ComputeCholesky,3,Cplfloat,((Cplfloat (*)[8
30  SMECY_launch(ComputeCholesky,3,STHORM,(threadprivate_i_F_0 % 4),0);
    SMECY_get_arg_vector(ComputeCholesky,3,Cplfloat,
32    ((Cplfloat (*)[8UL])((UG -> SegClusters)[i_F_0 % 4].S3.S4.F_R_BlockCho
      8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
34  SMECY_cleanup_send_arg_vector(ComputeCholesky,2,Cplfloat,
      ((Cplfloat (*)[8UL])((UG -> SegClusters)[i_F_0 % 4].S3.F_X_2_out[0][0]
36    8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
    SMECY_cleanup_send_arg(ComputeCholesky,1,int,8,STHORM,
38    (threadprivate_i_F_0 % 4),0);
    SMECY_accelerator_end(ComputeCholesky,5,STHORM,(threadprivate_i_F_0 % 4
40  [...]
```

- Rely heavily on C pre-processor to this code
- Hit some gcc 4.7 C pre-processor bug...

SME-C IR2 output from smecc (accelerator) (I)

```
1  /* Code to be executed on the accelerator */
2  void smecy_accel_ComputeCholesky_5(SMECY_accel_func_args) {
     /* On-stack storage to send/receive address parameters
4       to/from the host: */
     int *i_F_0p__;
6
     int *i_F_0 = (int *)i_F_0p__;
8    SMECY_set(ComputeCholesky,5,STHORM,(threadprivate_i_F_0 % 4),0);
     SMECY_send_arg(ComputeCholesky,1,int,8,STHORM,(threadprivate_i_F_0 % 4),0
10   SMECY_send_arg_vector(ComputeCholesky,2,Cplfloat,
       ((Cplfloat (*)[8UL])((UG -> SegClusters)[ *i_F_0 % 4].S3.F_X_2_out[0][0]
12     8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
     SMECY_prepare_get_arg_vector(ComputeCholesky,3,Cplfloat,
14     ((Cplfloat (*)[8UL])((UG -> SegClusters)[ *i_F_0 % 4].S3.S4.F_R_BlockCho
       8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
16   SMECY_launch(ComputeCholesky,3,STHORM,(threadprivate_i_F_0 % 4),0);
     SMECY_get_arg_vector(ComputeCholesky,3,Cplfloat,
18     ((Cplfloat (*)[8UL])((UG -> SegClusters)[ *i_F_0 % 4].S3.S4.F_R_BlockCho
       8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
20   SMECY_cleanup_send_arg_vector(ComputeCholesky,2,Cplfloat,
       ((Cplfloat (*)[8UL])((UG -> SegClusters)[ *i_F_0 % 4].S3.F_X_2_out[0][0]
22     8 * 8,STHORM,(threadprivate_i_F_0 % 4),0);
     SMECY_cleanup_send_arg(ComputeCholesky,1,int,8,STHORM,
24     (threadprivate_i_F_0 % 4),0);
```

SME-C IR2 output from smecc (accelerator) (II)

```
  SMECY_accelerator_end(ComputeCholesky,5,STHORM,(threadprivate_i_F_0 % 4),(
}

/* The dispatch loop on the accelerator side for one PE */
SMECY_begin_accel_function_dispatch
  SMECY_dispatch_accel_func(MatMultiply, 1)
  SMECY_dispatch_accel_func(MatMultiply, 2)
  SMECY_dispatch_accel_func(Subtract, 3)
  SMECY_dispatch_accel_func(DataReplace2, 4)
  SMECY_dispatch_accel_func(ComputeCholesky, 5)
  SMECY_dispatch_accel_func(Invert, 6)
  SMECY_dispatch_accel_func(Reconstruct2, 7)
SMECY_end_accel_function_dispatch

/* The hook to start the PEs */
SMECY_start_PEs_dispatch
```

IR2 expansion for STHORM PE fabric code (I)

(code for error checking removed)

```
1  void smecy_accel_ComputeCholesky_5(mcapi_pktchan_send_hndl_t P4A_transmit,
2                                     mcapi_pktchan_recv_hndl_t P4A_receive)
                                      {
4    [...]
     {
6      size_t P4A_received_size;
       int* p4a_STHORM_ComputeCholesky_1_msg;
8      mcapi_pktchan_recv(P4A_receive,
                          (void **)&p4a_STHORM_ComputeCholesky_1_msg,
10                        &P4A_received_size, &SMECY_MCAPI_status);
       int p4a_STHORM_ComputeCholesky_1 = *p4a_STHORM_ComputeCholesky_1_msg;
12     Cplfloat p4a_STHORM_ComputeCholesky_2[8 * 8];
       SMECY_MCAPI_receive(P4A_receive,
14                         p4a_STHORM_ComputeCholesky_2,
                           sizeof(p4a_STHORM_ComputeCholesky_2));
16     Cplfloat p4a_STHORM_ComputeCholesky_3[8 * 8];
       ComputeCholesky(p4a_STHORM_ComputeCholesky_1,
18                     p4a_STHORM_ComputeCholesky_2,
                       p4a_STHORM_ComputeCholesky_3);
20     SMECY_MCAPI_send(P4A_transmit,
                        p4a_STHORM_ComputeCholesky_3,
22                      8 * 8*sizeof(Cplfloat));
```

IR2 expansion for STHORM PE fabric code (///)

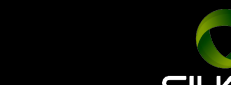
```
    }
24  }
```

Compiler organization (I)

- Use ROSE compiler infrastructure
- 2500 SLOC C++
- 500 SLOC Flex/Bison for the pragma
- 2000 SLOC C for runtime

```
1  void translateSmecy(SgProject *p) {
2    // Preprocessing of the project.
     attachAttributes(p);
4    parseExpressions(p);
     addSmecyInclude(p);
6
     // Translating the different kinds of pragmas
8    translateStreaming(p);
     translateMapping(p);
10 }

12   /* If directive has a map clause,
        translates it in corresponding calls to SMECY API
14    */
     void translateMap(SgStatement* target, Attribute* attribute,
16       SgStatement* functionToMap, int instance_number)
```



Compiler organization (II)

```
{
  // Deal with the case the function to map is used in a declaration
  processVariableDeclaration(target, attribute, functionToMap);
  // Get size information if not set in #pragma arg
  completeSizeInfo(target, attribute, functionToMap);
  // Deal with #pragma if(bool) for conditional remote execution
  processIf(target, attribute, functionToMap);

  // Deal with function parameters
  SgScopeStatement* scope = SageInterface::getScope(functionToMap);
  std::vector<SgExpression*> mapCoordinates = attribute->getMapCoordinates
  SgExpression* mapName = attribute->getMapName(scope);

  // Add calls to various SMECY API macros
  addSmecySet(target, mapName, mapCoordinates,
      getFunctionRef(functionToMap), instance_number);
  processArgs(target, attribute, functionToMap);
  addSmecyLaunch(target, mapName, mapCoordinates, functionToMap);
  processReturn(target, attribute, functionToMap);

  //removing pragma declaration TODO free memory
  SageInterface::removeStatement(target);
}
```

OpenCL target

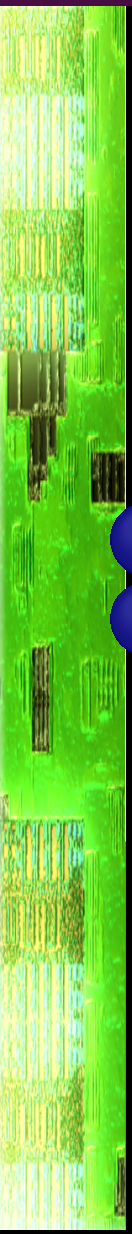
- Need to target OpenCL too for STHORM, GPU and others
- PIPS-based Par4All at SILKAN already generates OpenCL, but not for SME-C...
- `smecc` extracts functions with

```
1        #pragma smecy map(OpenCL)
```

  - Generate XML description of SME-C OpenCL parts
- Par4All operates on OpenCL-flagged functions
  - Loop fusion, scalarization...
  - Parallelization based on abstract interpretation
  - Kernel extraction with outlining
  - Communication generation based on memory-region analysis
  - Call to OpenCL API

SILKAN

Outline



SILKAN

Conclusion

- Definition of the SME-C `#pragma` extension
- Simple incremental model & language, OpenMP-based
- Also an Internal Representation for tool interconnection!
- ⇝ `smecc` compiler prototype
  `https://github.com/silkan/smecc`
- Experience with `#pragma` language at SILKAN
- Experience with source-to-source ROSE Compiler & C++11 at SILKAN
- New recyclable back-end for a new class of target
  - Generic programming model for heterogeneous computing, Tutor/Arion bus, hardware accelerators, MPI, Cloud... ☺ according to parallelism grain size ☹
- Prototype to be ported to Clang/LLVM in Par4All 2.x
- Need interprocedural data-flow analysis to optimize data transfers