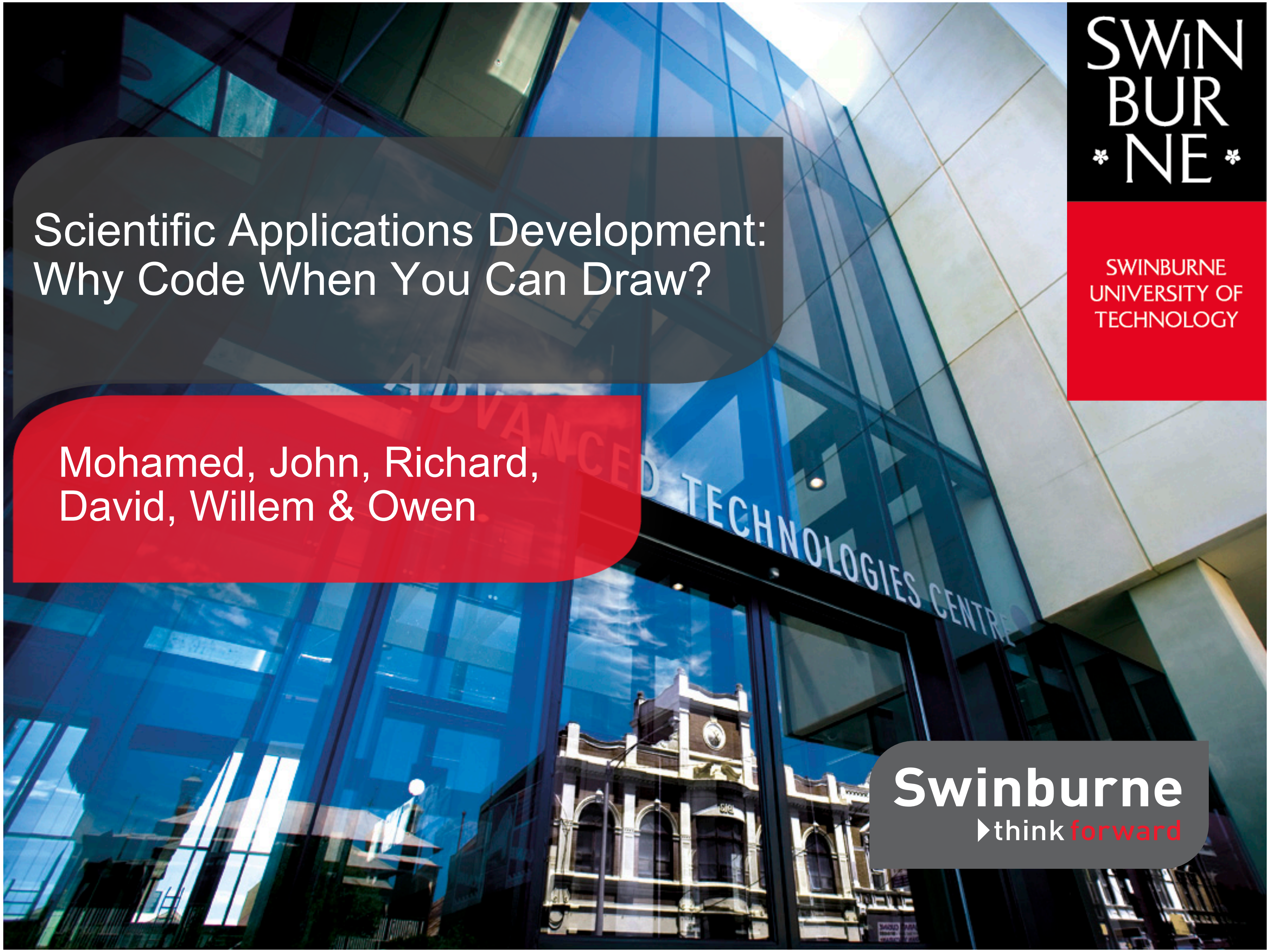


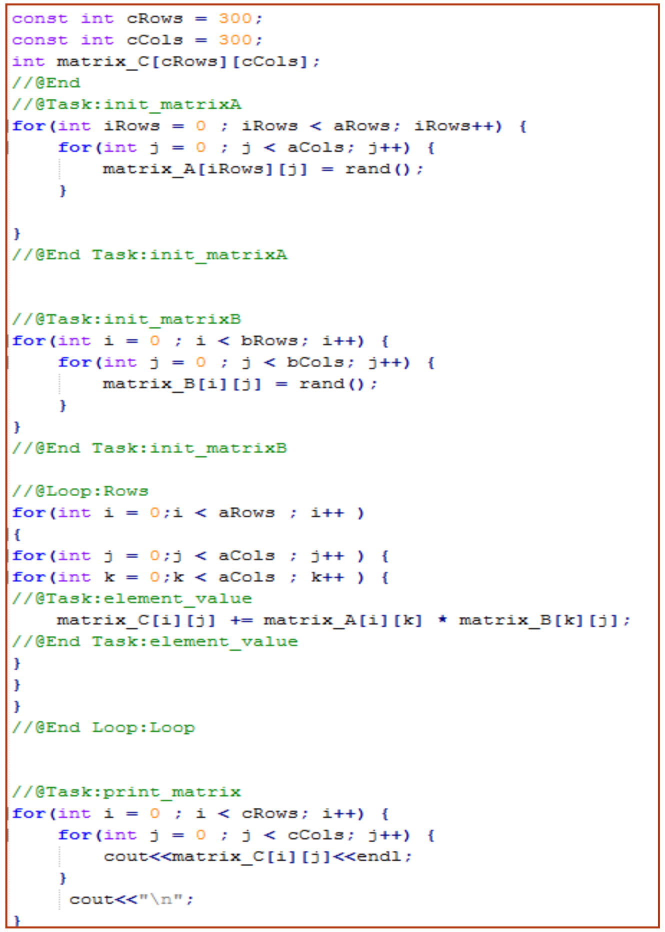
Scientific Applications Development: Why Code When You Can Draw?

Mohamed, John, Richard, David, Willem & Owen

SWINBURNE UNIVERSITY OF TECHNOLOGY

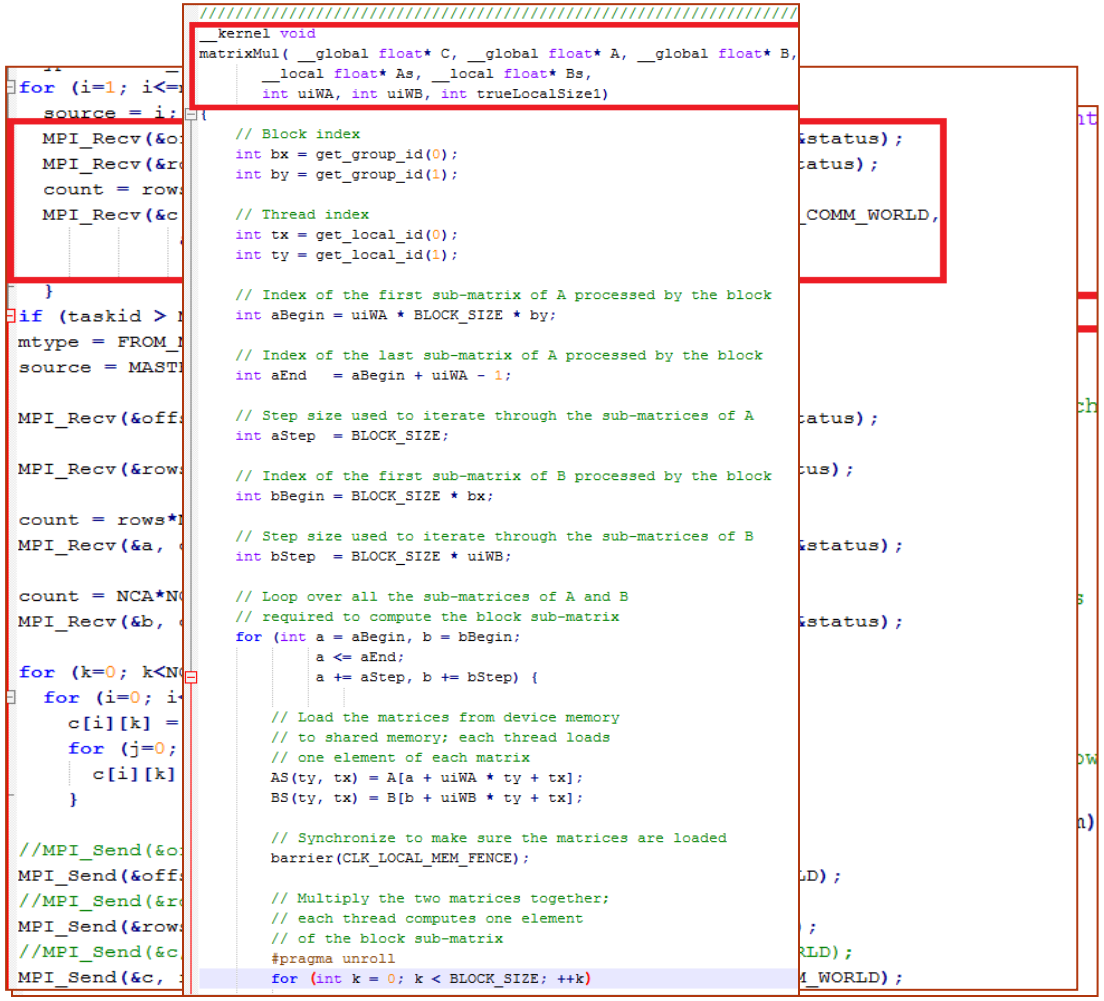
Swinburne think forward

Objective



```
const int cRows = 300;
const int cCols = 300;
int matrix_C[cRows][cCols];
//@End
//@Task:init_matrixA
for(int iRows = 0 ; iRows < aRows; iRows++) {
    for(int j = 0 ; j < aCols; j++) {
        matrix_A[iRows][j] = rand();
    }

}
//@End Task:init_matrixA


//@Task:init_matrixB
for(int i = 0 ; i < bRows; i++) {
    for(int j = 0 ; j < bCols; j++) {
        matrix_B[i][j] = rand();
    }
}
//@End Task:init_matrixB

//@Loop:Rows
for(int i = 0;i < aRows ; i++ )
{
for(int j = 0;j < aCols ; j++ ) {
for(int k = 0;k < aCols ; k++ ) {
//@Task:element_value
    matrix_C[i][j] += matrix_A[i][k] * matrix_B[k][j];
//@End Task:element_value
}
}
}
//@End Loop:Loop


//@Task:print_matrix
for(int i = 0 ; i < cRows; i++) {
    for(int j = 0 ; j < cCols; j++) {
        cout<<matrix_C[i][j]<<endl;
    }
   cout<<"\n";
}
```

Sequential Matrix Multiplication



```
for (i=1; i<=
  source = i;
  MPI_Recv(&o
  MPI_Recv(&r
  count = row
  MPI_Recv(&c

  }
if (taskid > M
mtype = FROM_M
source = MAST

MPI_Recv(&off

MPI_Recv(&row

count = rows*N
MPI_Recv(&a, 

count = NCA*N
MPI_Recv(&b, 

for (k=0; k<N
  for (i=0; i
    c[i][k] =
    for (j=0;
      c[i][k]
    }

//MPI_Send(&o
MPI_Send(&off
//MPI_Send(&r
MPI_Send(&row
//MPI_Send(&c
MPI_Send(&c, 
```

```
////////////////////////////////////////////////////////////////
__kernel void
matrixMul( __global float* C, __global float* A, __global float* B,
        __local float* As, __local float* Bs,
        int uiWA, int uiWB, int trueLocalSize1)
{
    // Block index
    int bx = get_group_id(0);
    int by = get_group_id(1);

    // Thread index
    int tx = get_local_id(0);
    int ty = get_local_id(1);

    // Index of the first sub-matrix of A processed by the block
    int aBegin = uiWA * BLOCK_SIZE * by;

    // Index of the last sub-matrix of A processed by the block
    int aEnd   = aBegin + uiWA - 1;

    // Step size used to iterate through the sub-matrices of A
    int aStep  = BLOCK_SIZE;

    // Index of the first sub-matrix of B processed by the block
    int bBegin = BLOCK_SIZE * bx;

    // Step size used to iterate through the sub-matrices of B
    int bStep  = BLOCK_SIZE * uiWB;

    // Loop over all the sub-matrices of A and B
    // required to compute the block sub-matrix
    for (int a = aBegin, b = bBegin;
             a <= aEnd;
             a += aStep, b += bStep) {

        // Load the matrices from device memory
        // to shared memory; each thread loads
        // one element of each matrix
        AS(ty, tx) = A[a + uiWA * ty + tx];
        BS(ty, tx) = B[b + uiWB * ty + tx];

        // Synchronize to make sure the matrices are loaded
        barrier(CLK_LOCAL_MEM_FENCE);

        // Multiply the two matrices together;
        // each thread computes one element
        // of the block sub-matrix
        #pragma unroll
        for (int k = 0; k < BLOCK_SIZE; ++k)
```

```
&status);
tatus);

_COMM_WORLD,

tatus);

tus);

&status);

&status);

LD);

);
RLD);
M_WORLD);
```

Matrix Multiplication - CUDA

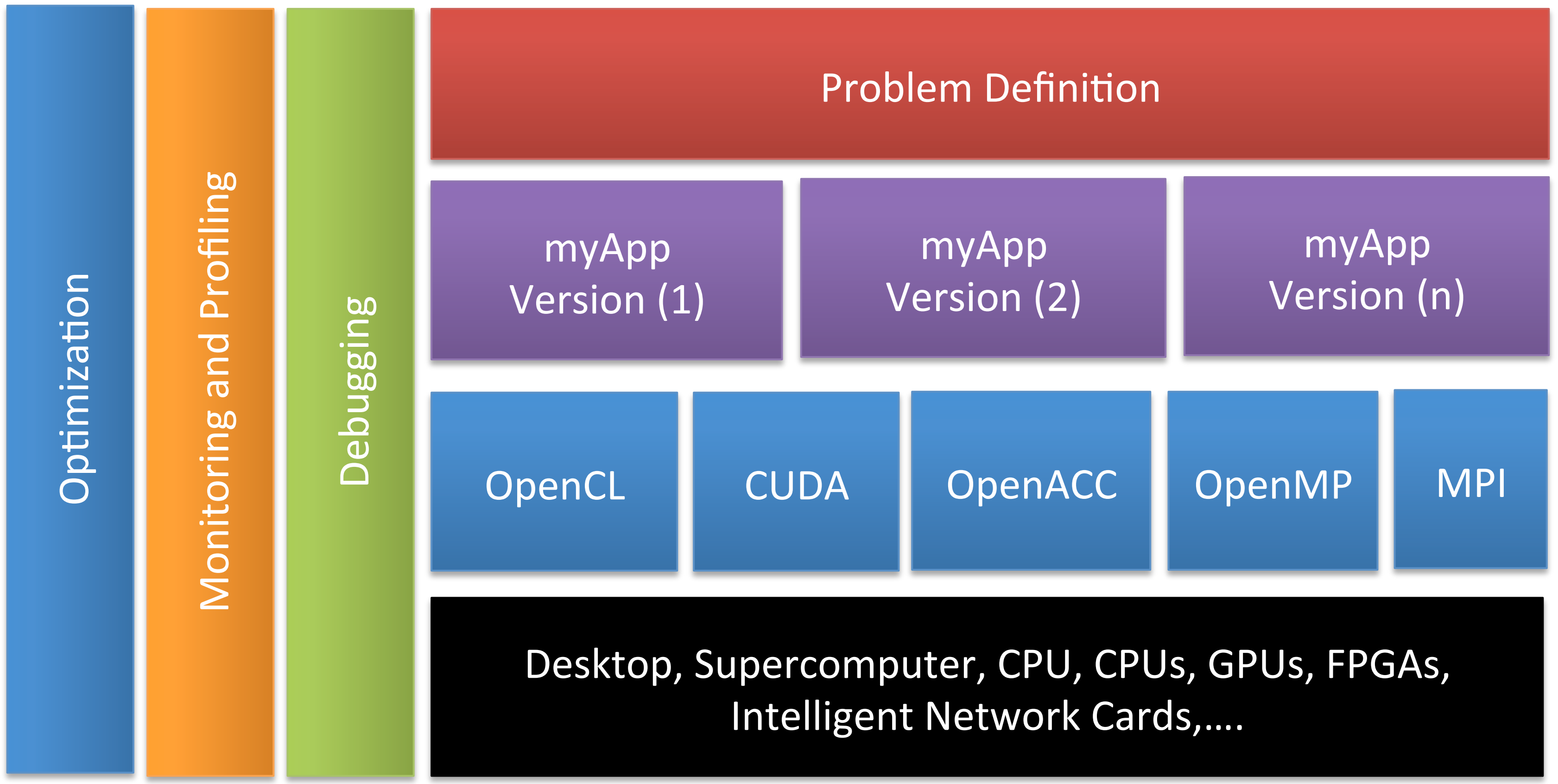
Outline

- Problem / Motivation
- Approach
- Example Usage
- Future work

Problem

- Scientists are increasingly developing complex software for data analysis
- *"It's all about the software…"*
- Most are not trained programmers
- Many are using complex software platforms and techniques e.g. distributed & parallel programming, GPUs, etc - that are hard for experienced CS grads to do
- Approaches to address range from packaged software (Lack flexibility), DSLs (Also Flexibility/Domain issues), programming patterns and toolkits (still complex)

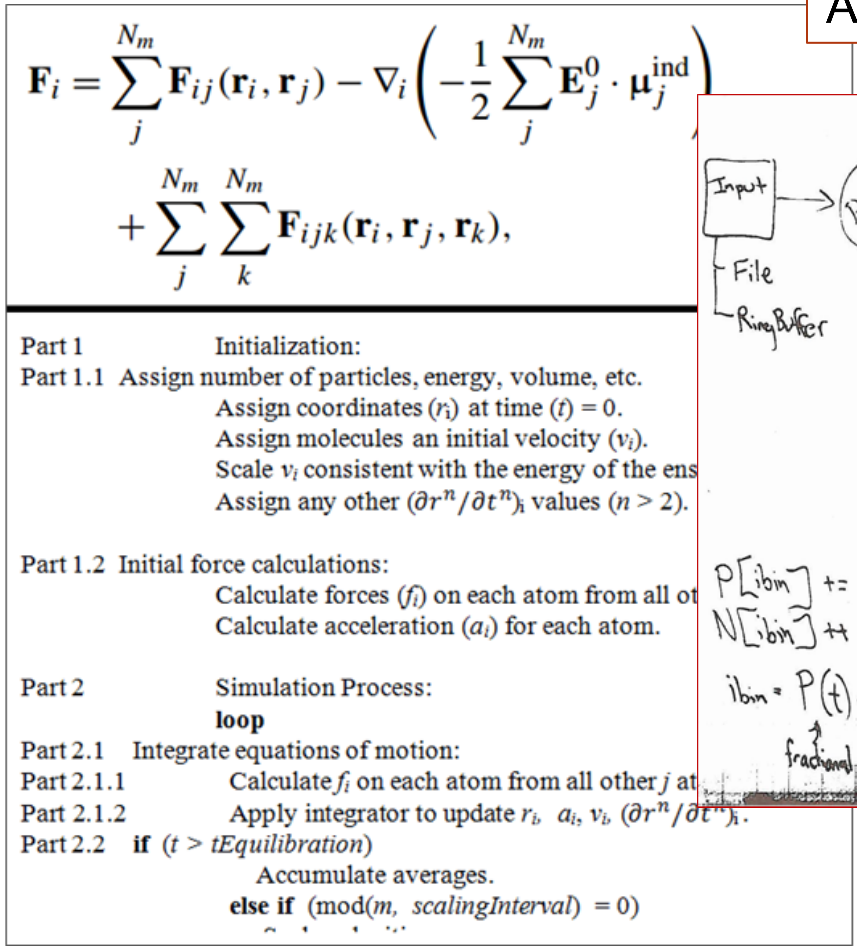
More Problems…



Lack of high-level, human-centric approach to help in developing high quality scientific apps

Motivation – How scientists design their applications…

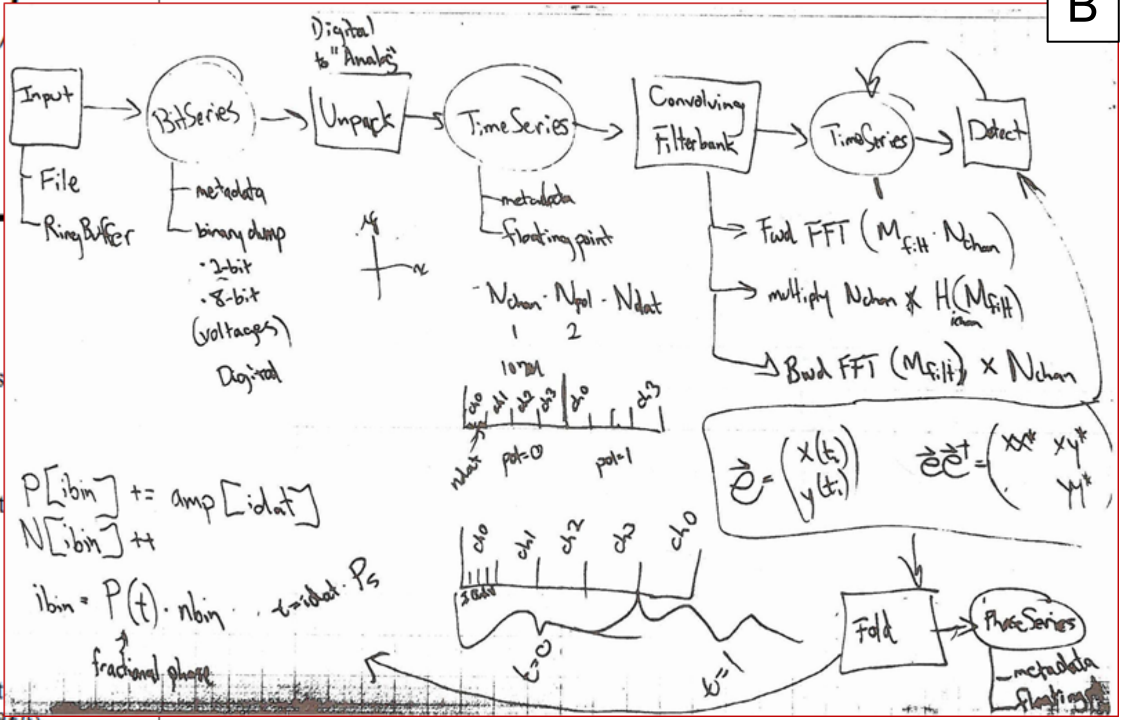
A



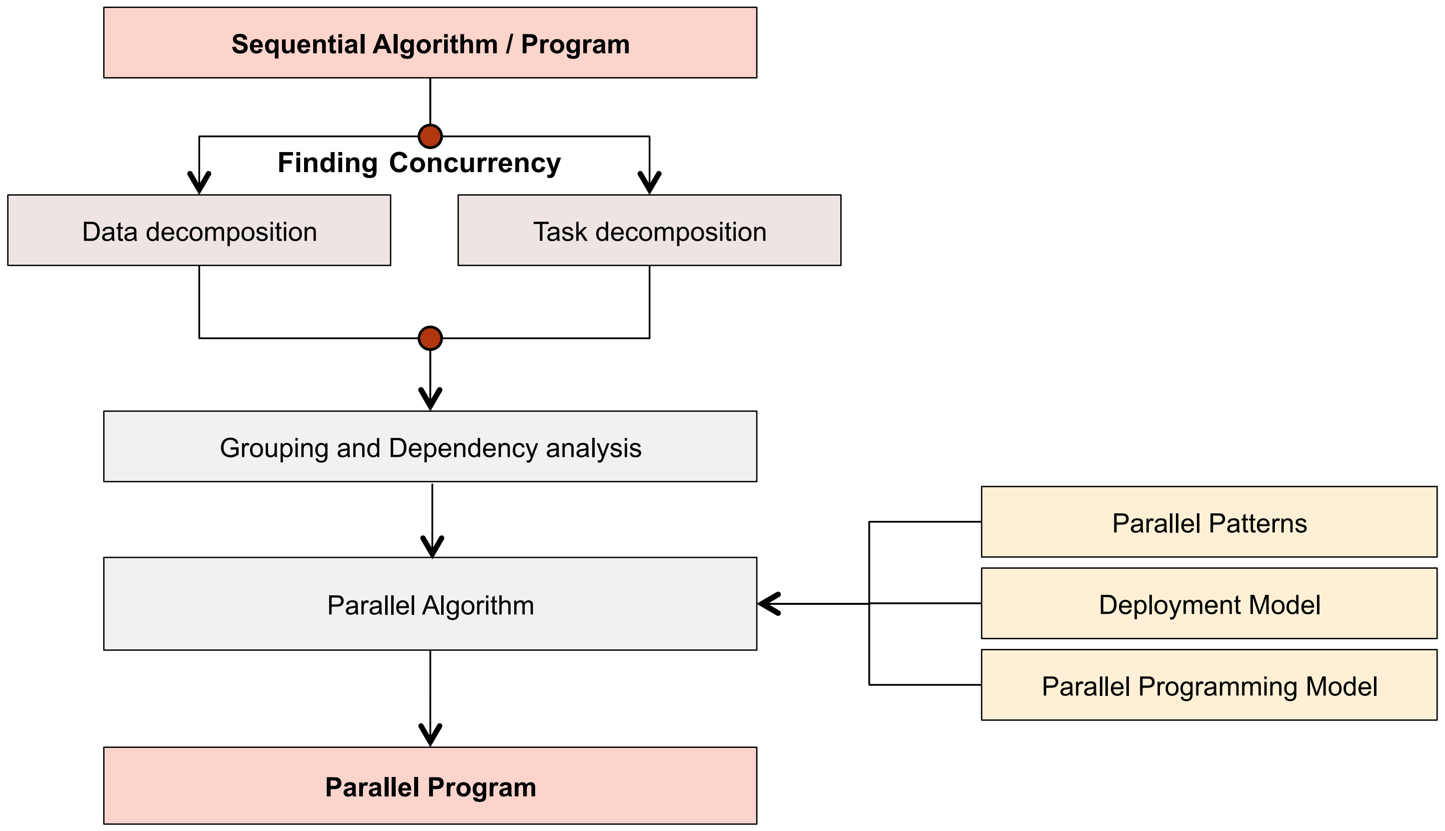
$$\mathbf{F}_i = \sum_j^{N_m} \mathbf{F}_{ij}(\mathbf{r}_i,\mathbf{r}_j) - \nabla_i\left(-\frac{1}{2}\sum_j^{N_m}\mathbf{E}_j^0\cdot\mu_j^{\text{ind}}\right)$$
$$+\sum_j^{N_m}\sum_k^{N_m}\mathbf{F}_{ijk}(\mathbf{r}_i,\mathbf{r}_j,\mathbf{r}_k),$$

```
Part 1              Initialization:
Part 1.1  Assign number of particles, energy, volume, etc.
              Assign coordinates ($r_i$) at time ($t$) = 0.
              Assign molecules an initial velocity ($v_i$).
              Scale $v_i$ consistent with the energy of the ens
              Assign any other $(\partial r^n/\partial t^n)_i$ values ($n > 2$).

Part 1.2  Initial force calculations:
              Calculate forces ($f_i$) on each atom from all ot
              Calculate acceleration ($a_i$) for each atom.

Part 2              Simulation Process:
                    loop
Part 2.1    Integrate equations of motion:
Part 2.1.1        Calculate $f_i$ on each atom from all other $j$ at
Part 2.1.2        Apply integrator to update $r_i$, $a_i$, $v_i$, $(\partial r^n/\partial t^n)_i$.
Part 2.2    if ($t$ > tEquilibration)
                      Accumulate averages.
                  else if (mod($m$, scalingInterval) = 0)
```

B

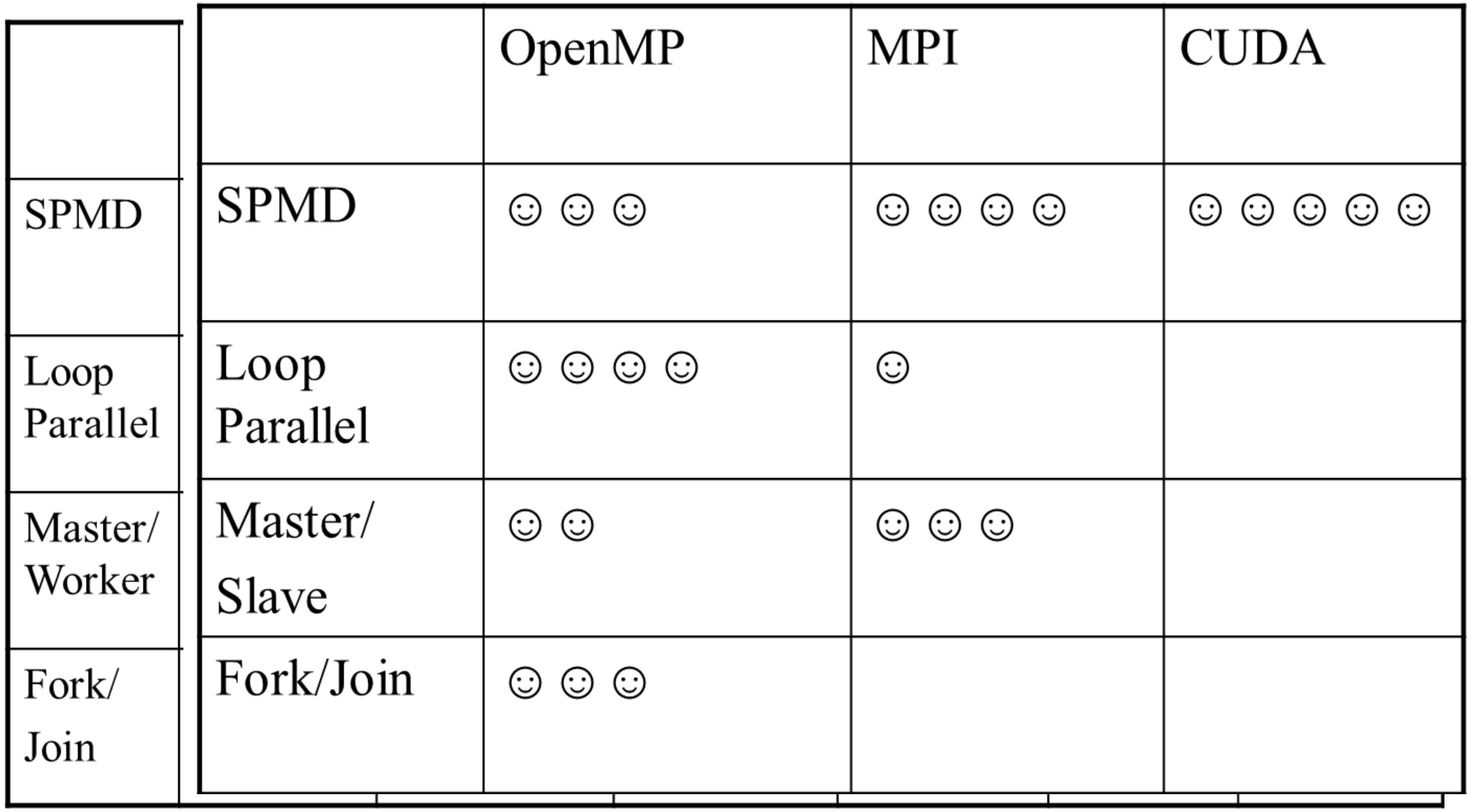


Parallel Program Development Steps



* Timothy. G. Mattson et al., 2004, Patterns for Parallel Programming, Addison Wesley Software Patterns Series.

Program Organization vs Parallel Patterns vs Parallel Programming Models



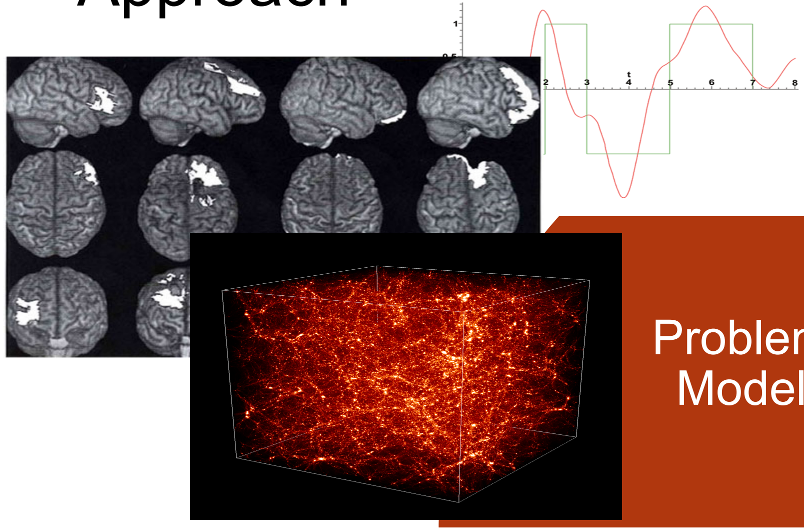
SPMD
Loop Parallel
Master/ Worker
Fork/ Join

	OpenMP	MPI	CUDA
SPMD	☺ ☺ ☺	☺ ☺ ☺ ☺	☺ ☺ ☺ ☺ ☺
Loop Parallel	☺ ☺ ☺ ☺	☺	
Master/ Slave	☺ ☺	☺ ☺ ☺	
Fork/Join	☺ ☺ ☺		

* Timothy. G. Mattson et al., 2004, Patterns for Parallel Programming, Addison Wesley Software Patterns Series.

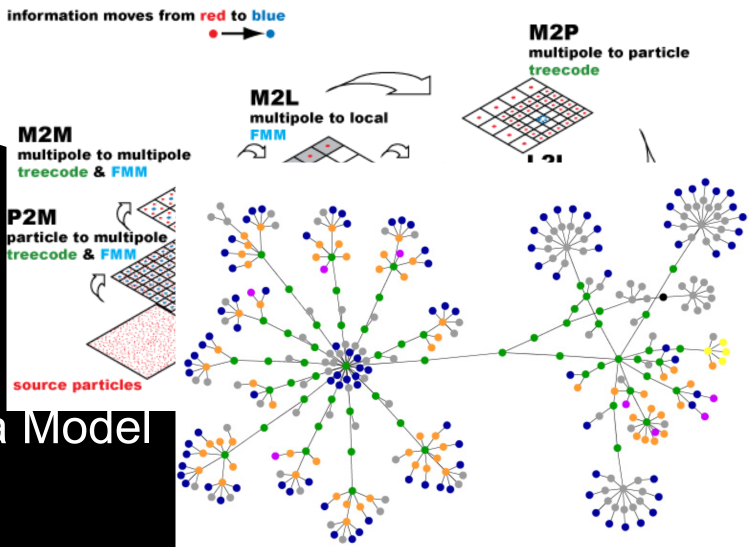
Swinburne

Approach



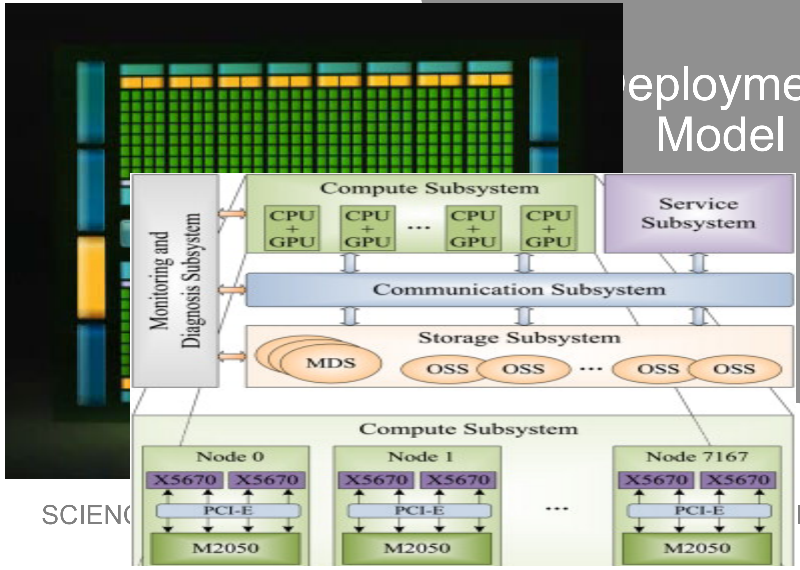
Problem Model

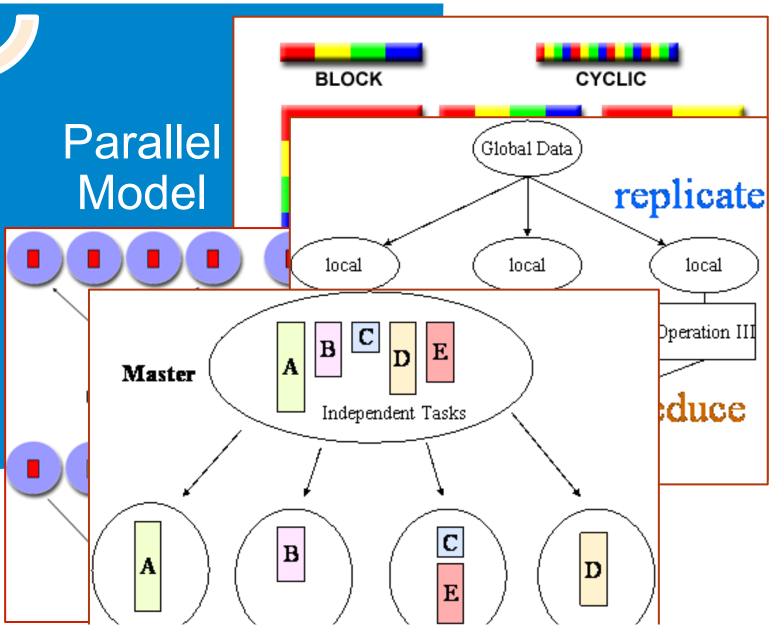
Data Model



eployment Model

Parallel Model





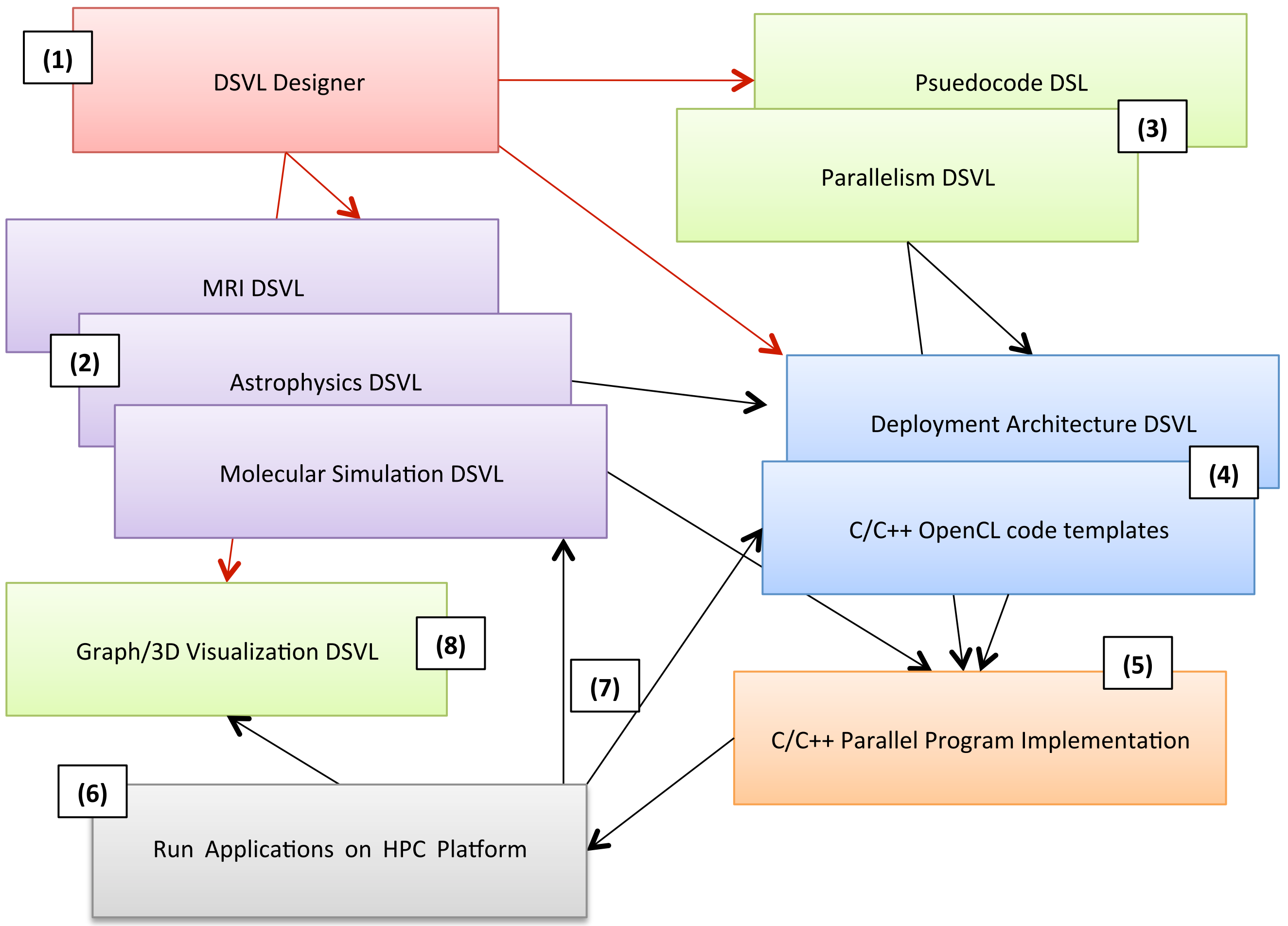
SCIENC

DESIGN

Approach

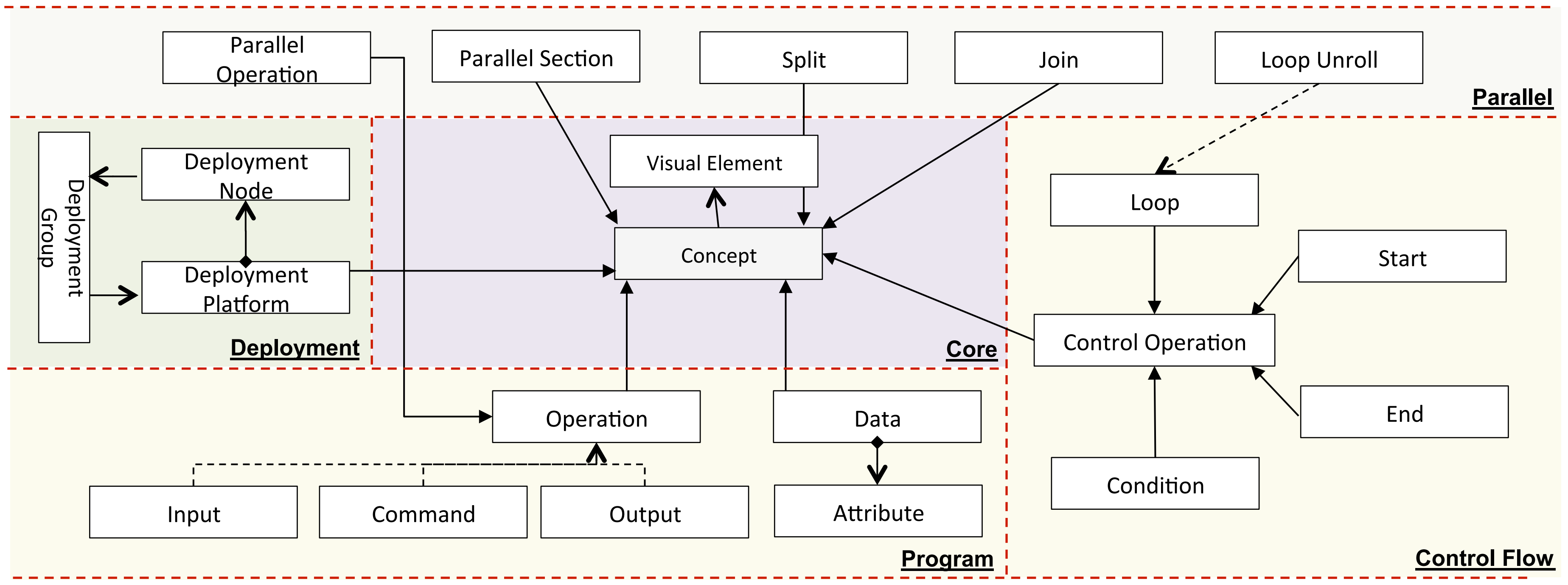
- Support scientists – and developers! – to model their applications at multiple levels of abstraction – domain right down to detailed C/C++/GPU kernel code
- Use set of user-defined and reusable DSVLs to model
- Provide web-based envrionment including DSVLs designer, coding, debugging, linking DSVL views
- Provide semi-automated support for generating lower-level models, generate code/code annotations, reverse-engineer higher-level models from (existing) code

Approach

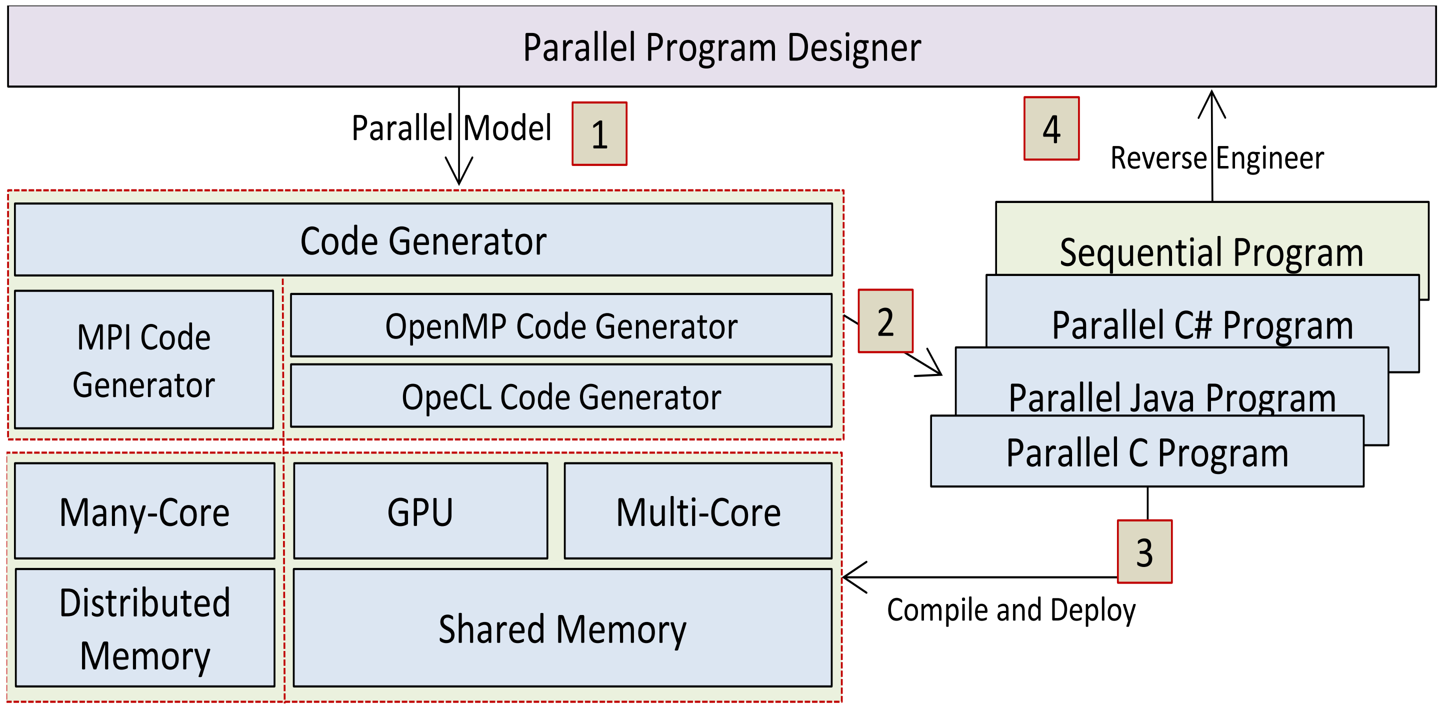

(1)
DSVL Designer
Psuedocode DSL
(3)
Parallelism DSVL
MRI DSVL
(2)
Astrophysics DSVL
Molecular Simulation DSVL
Deployment Architecture DSVL
(4)
C/C++ OpenCL code templates
Graph/3D Visualization DSVL
(8)
(7)
(5)
C/C++ Parallel Program Implementation
(6)
Run Applications on HPC Platform

*DSVL = Domain-Specific Visual Language

DSVL Designer Meta-model

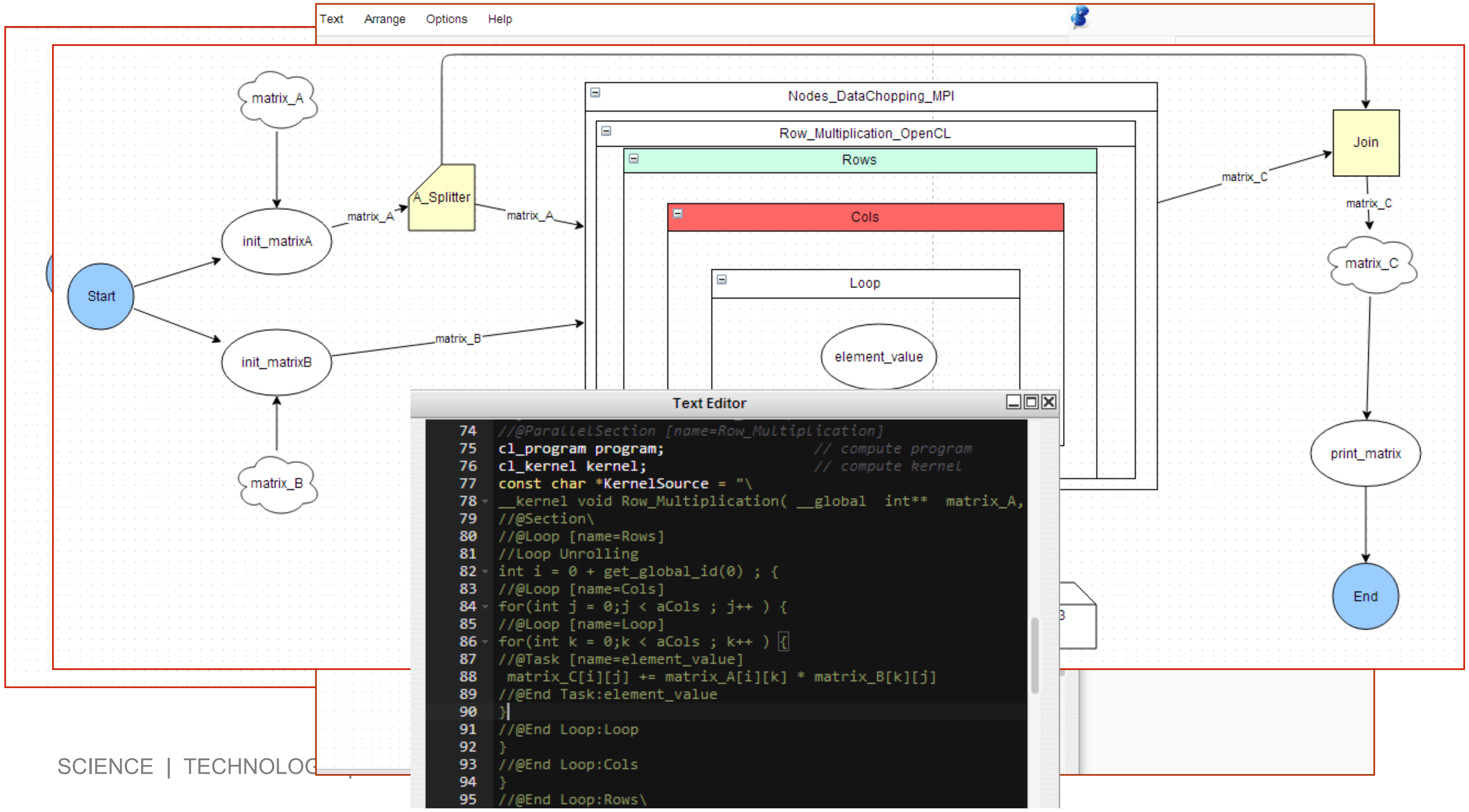


Platform


Parallel Program Designer
Parallel Model
1
4
Reverse Engineer
Code Generator
MPI Code Generator
OpenMP Code Generator
OpeCL Code Generator
2
Sequential Program
Parallel C# Program
Parallel Java Program
Parallel C Program
Many-Core
GPU
Multi-Core
Distributed Memory
Shared Memory
3
Compile and Deploy

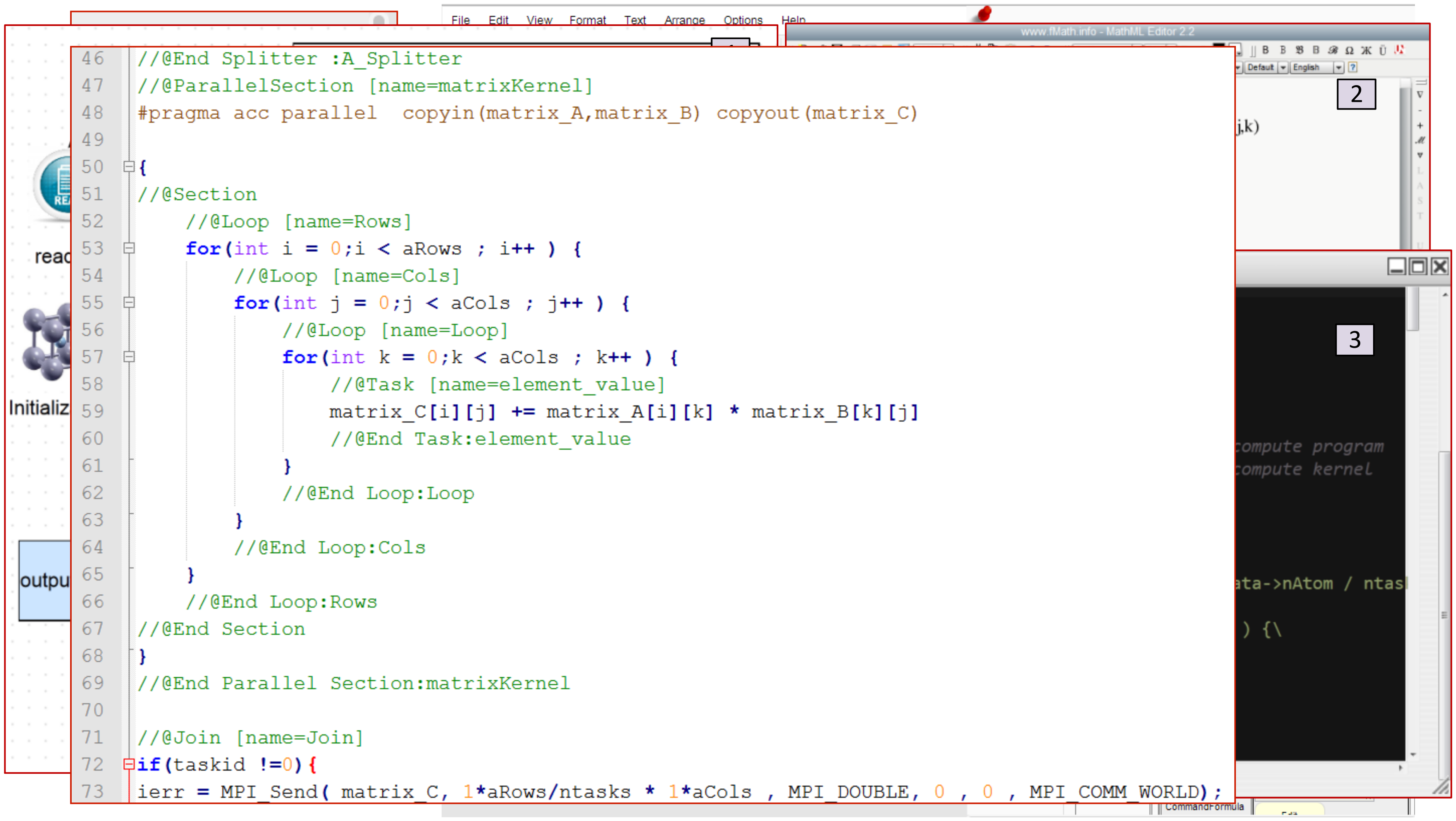
Example – Matrix Multiplication

$$C = A \times B \ , \ C_{i,j} = \mathrm{Sum}(\ A_{i,k} * B_{k,j}\ )$$



```
//@ParallelSection [name=Row_Multiplication]
cl_program program;                    // compute program
cl_kernel kernel;                      // compute kernel
const char *KernelSource = "\
__kernel void Row_Multiplication( __global  int**  matrix_A,
//@Section\
//@Loop [name=Rows]
//Loop Unrolling
int i = 0 + get_global_id(0) ; {
//@Loop [name=Cols]
for(int j = 0;j < aCols ; j++ ) {
//@Loop [name=Loop]
for(int k = 0;k < aCols ; k++ ) {
//@Task [name=element_value]
 matrix_C[i][j] += matrix_A[i][k] * matrix_B[k][j]
//@End Task:element_value
}
//@End Loop:Loop
}
//@End Loop:Cols
}
//@End Loop:Rows\
```

Snapshots



```
//@End Splitter :A_Splitter
//@ParallelSection [name=matrixKernel]
#pragma acc parallel  copyin(matrix_A,matrix_B) copyout(matrix_C)

{
//@Section
    //@Loop [name=Rows]
    for(int i = 0;i < aRows ; i++ ) {
        //@Loop [name=Cols]
        for(int j = 0;j < aCols ; j++ ) {
            //@Loop [name=Loop]
            for(int k = 0;k < aCols ; k++ ) {
                //@Task [name=element_value]
                matrix_C[i][j] += matrix_A[i][k] * matrix_B[k][j]
                //@End Task:element_value
            }
            //@End Loop:Loop
        }
        //@End Loop:Cols
    }
    //@End Loop:Rows
//@End Section
}
//@End Parallel Section:matrixKernel

//@Join [name=Join]
if(taskid !=0){
ierr = MPI_Send( matrix_C, 1*aRows/ntasks * 1*aCols , MPI_DOUBLE, 0 , 0 , MPI_COMM_WORLD);
```

```
j,k)
```

```
compute program
compute kernel

ata->nAtom / ntas
) {\
```

Summary & Future work

- Integrated web-based development environment for scientific applications
- Flexible DSVL designer with pre-packaged DSVLs (Parallel DSVL, and Deployment DSVL) and user-defined DSVLs
- Semi-automated roundtrip engineering support: model-> code-> model
- Working on:
  - Template-based DSVL Designer
  - Patterns and critics to guide users, analyze models/code
  - Visualization of running parallel code onto models

SWIN BUR NE

SWINBURNE UNIVERSITY OF TECHNOLOGY

Questions?

Mohamed Almorsy

malmorsy@swin.edu.au

Thanks to ARC Discovery Project scheme for support…

Australian Government

Australian Research Council

Swinburne

think forward

References

Almorsy, M., Grundy, J.C. and Ruegg, U. HorusCML: Context-aware Domain Specific Visual Languages Designer, 2014 IEEE Symposium on Visual Languages and Human-Centric Computing, Melbourne, Australia, July 27-1 Aug 2014, IEEE CPS

Almorsy, M., Grundy, J.C., Sadus, R., Barnes, D., Kaluza, O., van Straten, W., A Suite of Domain-Specific Visual Languages For Scientific Software Application Modelling, 2013 IEEE Symposium on Visual Languages and Human-Centric Computing, San Jose, CA, USA, Sept 15-19 2013, IEEE CPS.

Almorsy, M., Grundy, J.C., Sadus, R., Barnes, D., Kaluza, O., van Straten, W., A Suite of Domain-Specific Visual Languages For Scientific Software Application Modelling, 2013 IEEE Symposium on Visual Languages and Human-Centric Computing, San Jose, CA, USA, Sept 15-19 2013, IEEE CPS.

Mairhofer, J. and Sadus, R.J., Thermodynamic properties of supercritical n-m Lennard-Jones fluids and Isochoric and Isobaric heat capacity maxima and minima. The Journal of Chemical Physics, 139, 154503 (2013).

Owen L Kaluza, Amanda C L Ng, David K Wright, Leigh A Johnston, John Grundy, and David G Barnes, Fast diffusion-guided QSM using Graphical Processing Units, Presentation at 21st Annual Meeting of the International Society for Magnetic Resonance in Medicine, Salt Lake City, Utah, April 2013.

Owen L. Kaluza, Amanda C. L. Ng, David G. Barnes, John Grundy, Accelerating Diffusion-guided Quantitative Susceptibility Mapping with OpenCL, Presentation at OzViz Accelerated Computing Workshop 2012, Perth, Australia, December 2012.