Exploring task decomposition strategies with Tareador

Eduard Ayguadé and Rosa M. Badia

Barcelona. November 2012

Part I

Tareador environment

Outline

Motivation

API and example of use

Installation

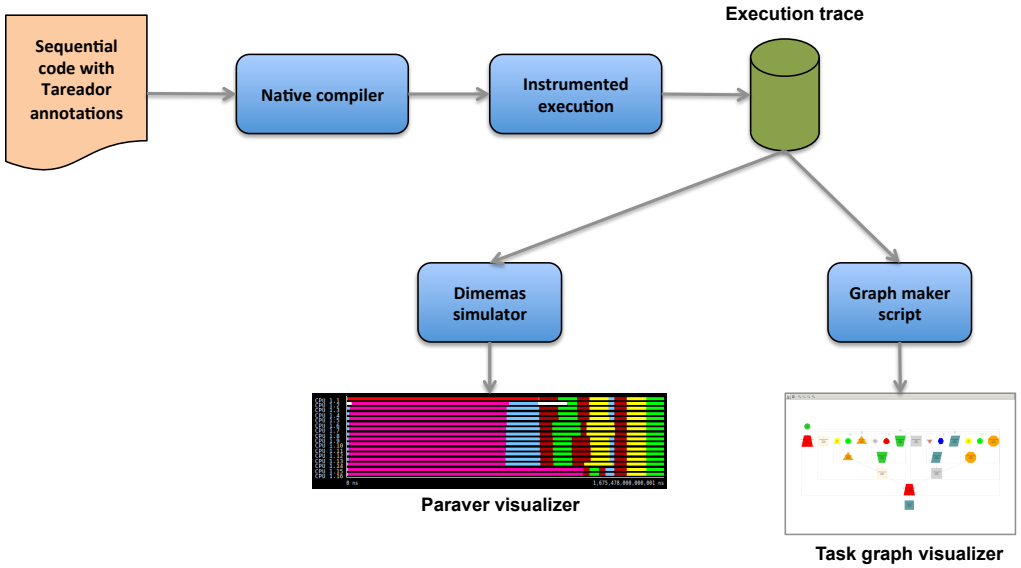
Parallelization strategy

- Task decomposition
  - From a sequential specification of the program ...
  - ... find a decomposition of the problem in tasks (i.e. identify pieces of work that can execute concurrently) ...
  - ... ensuring that the same result is produced (i.e. identify dependencies that impose ordering and data sharing constraints).
- These tasks and constraints can be later mapped to the elements offered by parallel programming languages

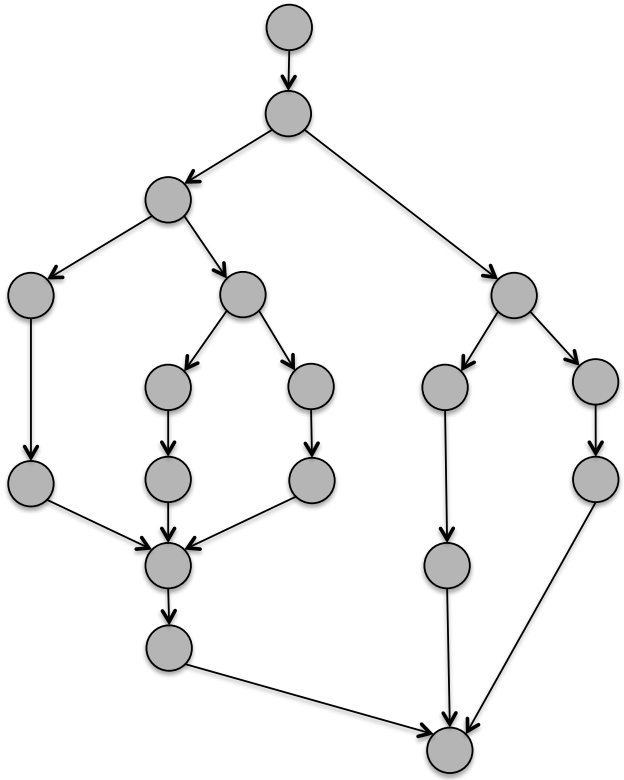
Guiding the task decomposition process

Tareador: environment to support the task decomposition process:

- API to annotate sequential program with potential tasks
- Binary instrumentation using a new **Valgrind** module
- Visualization of task graph (granularities and dependences)
- Simulation with **Dimemas** and visualization with **Paraver**

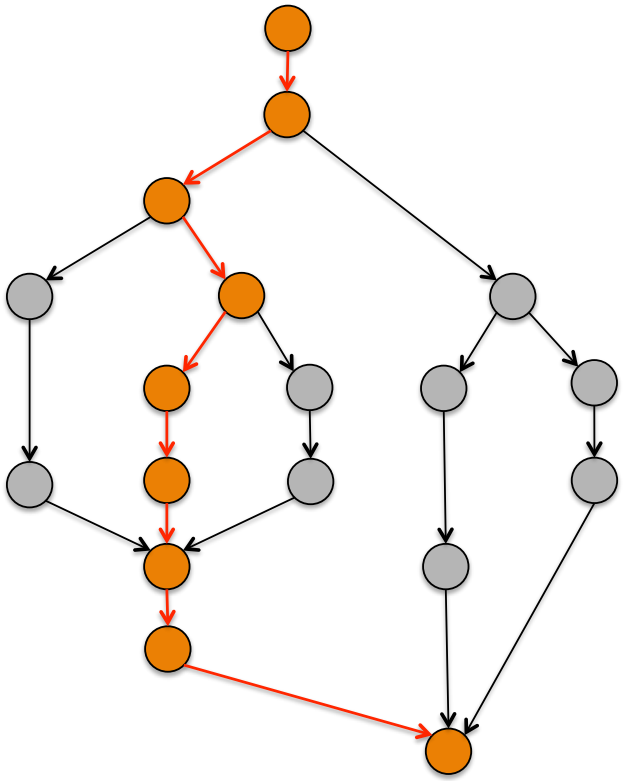


Computation task graph abstraction



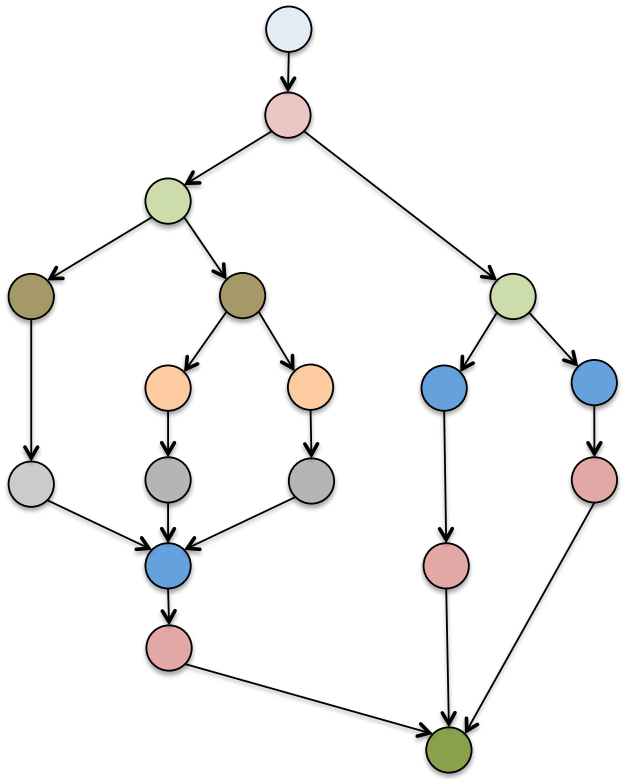
- Directed Acyclic Graph
- Node = dynamic instance of an annotated task (tracking of task entry/exit)
- Edge = dependence between tasks (tracking of dynamic allocations and memory accesses)

Computation task graph abstraction



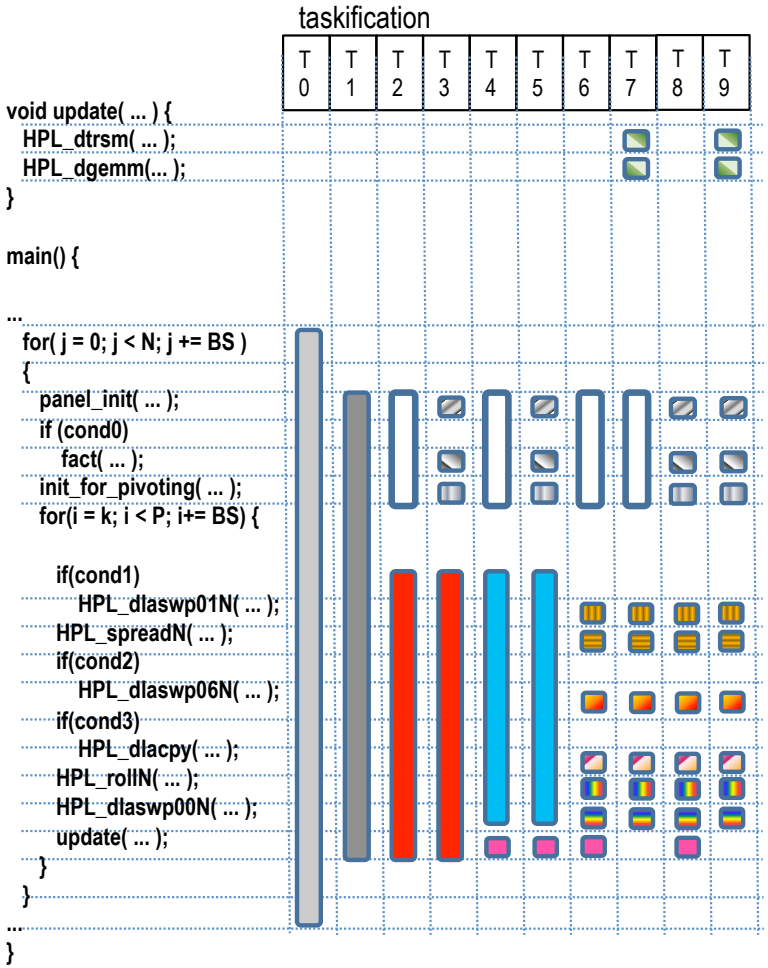
- $T_1 = \sum_{i=1}^{nodes}(work_node_i)$
- $T_\infty = \sum_{i \in criticalpath}(work_node_i)$, assuming sufficient resources
- Parallelism $= T_1/T_\infty$

Computation task graph abstraction



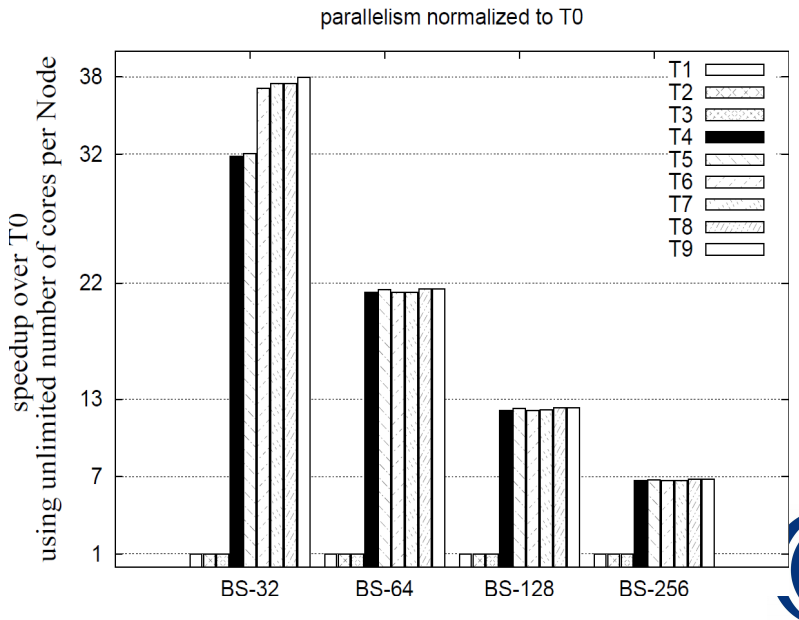
- T_p = execution time on P processors (depends on the schedule of the graph nodes on the processors)
- Speedup on P processors: $S_p = T_1/T_p$
- **Dimemas** reconstructs a temporal parallel execution to determine T_p:
  - parametrizable architecture
  - what if optimizing a certain task

Exploration of potential task decompositions



*HP Linpack:
4 MPI processes;
Problem size: 8192;
Block size: 256,128, 64, 32.

The parallelism is released in transition T3 -> T4 when function *update* is separated from the rest of the inner loop



Outline

Motivation

API and example of use

Installation

Tareador API

- Specification of tareador region

```
tareador_ON();
     ...
tareador_OFF();
```

- Specification of task boundaries

```
tareador_start_task("name of task");
     /* Code region / task */
tareador_end_task();
```

Recursive tasks are possible (e.g. divide and conquer approaches).

- Filtering objects

```
tareador_disable_object(address of object);
     /* Code region */
tareador_enable_object(address of object);
```

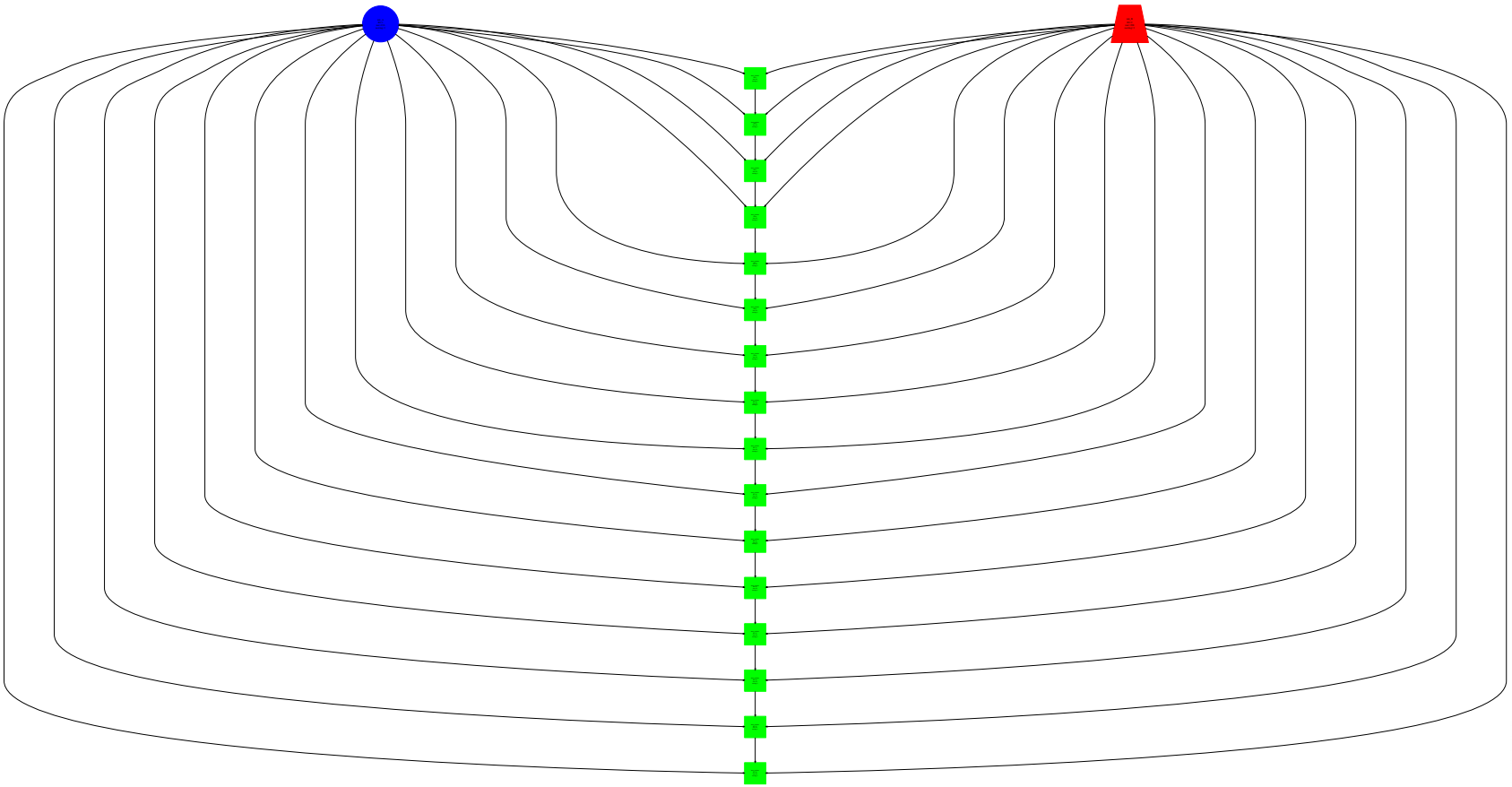
Example: dot product

Instrumented source code for iterative dot product:

```
void dot_product (long N,
    double A[N], double B[N], double *acc){
    double prod;

    *acc=0.0;
    for (int i=0; i<N; i++) {
        tareador_start_task("inner_product");
        prod = A[i]*B[i];
        *acc+= prod;
        tareador_end_task();
    }
}
```

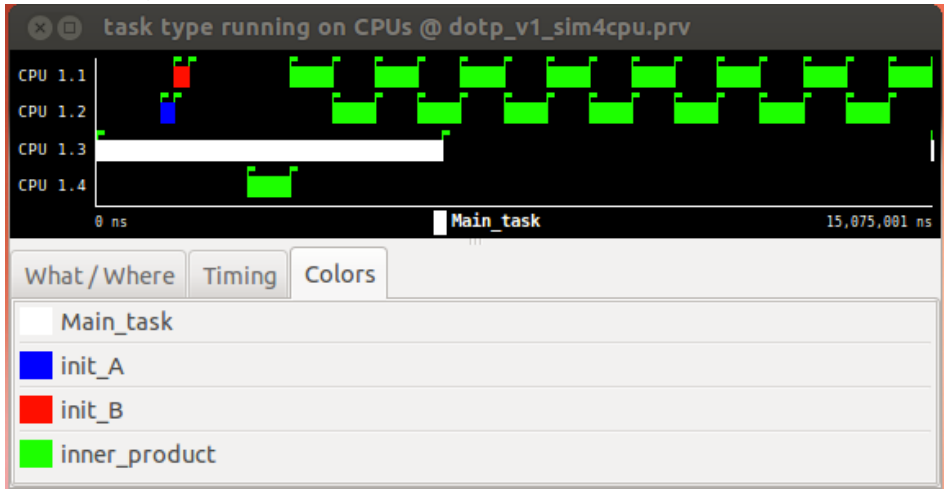
```
int main() {
    tareador_ON ();

    tareador_start_task("init_A");
    for (int i=0; i< N; i++) A[i]=i;
    tareador_end_task();

    tareador_start_task("init_B");
    for (int i=0; i< N; i++) B[i]=2*i;
    tareador_end_task();

    dot_product (N, A, B, &result);

    tareador_OFF ();

}
```

Dot product: task graph, N=16

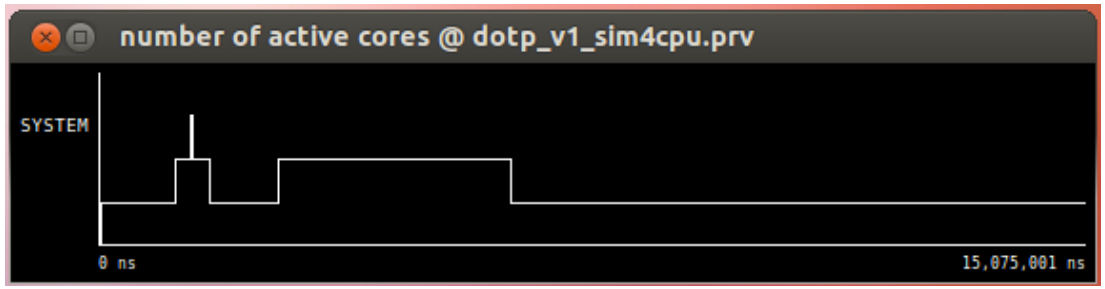


Dot product: Dimemas simulation, N=16, 4 CPU

- CPU view load configuration



- Maximum concurrency view configuration



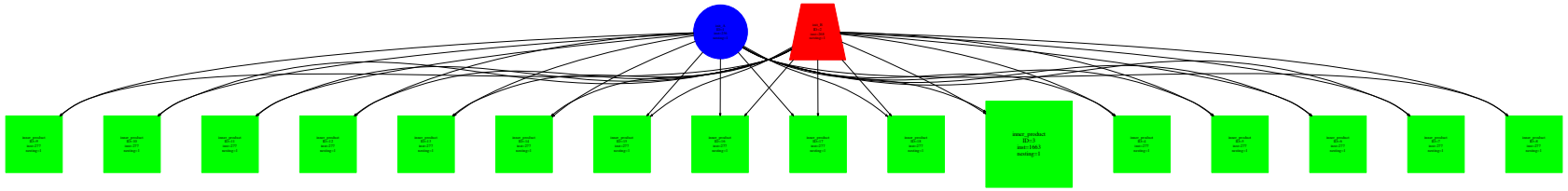
Example: dot product (cont.)

Instrumented source code (filtering object *acc*) for iterative dot product:

```
void dot_product (long N,
    double A[N], double B[N], double *acc){
    double prod;

    *acc=0.0;
    for (int i=0; i<N; i++) {
        tareador_start_task("inner_product");
        prod = A[i]*B[i];
        tareador_disable_object(acc);
        *acc+= prod;
        tareador_enable_object(acc);
        tareador_end_task();
    }
}
```

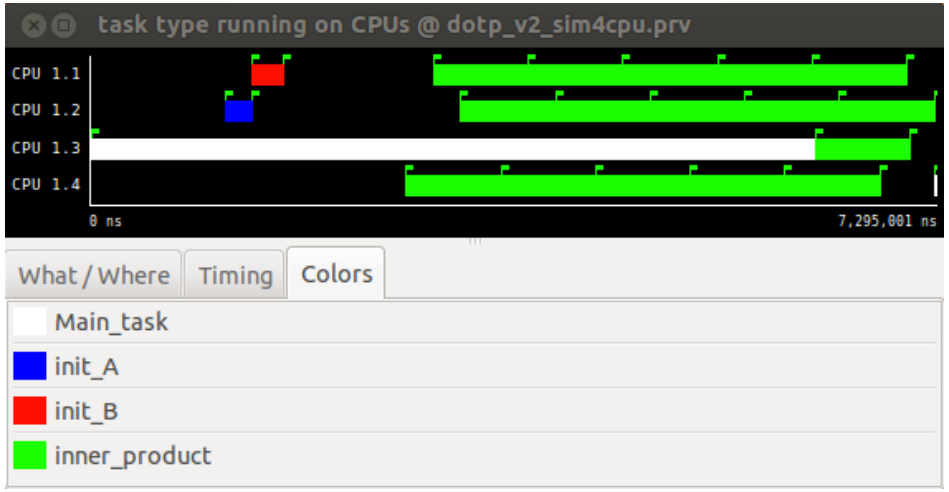
```
int main() {
    tareador_ON ();

    tareador_start_task("init_A");
    for (int i=0; i< N; i++) A[i]=i;
    tareador_end_task();

    tareador_start_task("init_B");
    for (int i=0; i< N; i++) B[i]=2*i;
    tareador_end_task();

    dot_product (N, A, B, &result);

    tareador_OFF ();
}
```

Dot product: task graph filtering *acc*, N=16

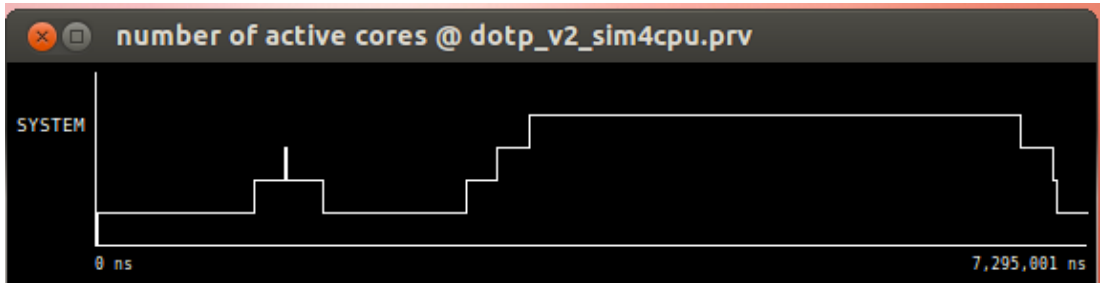


Dot product: Dimemas simulation filtering *acc*, N=16

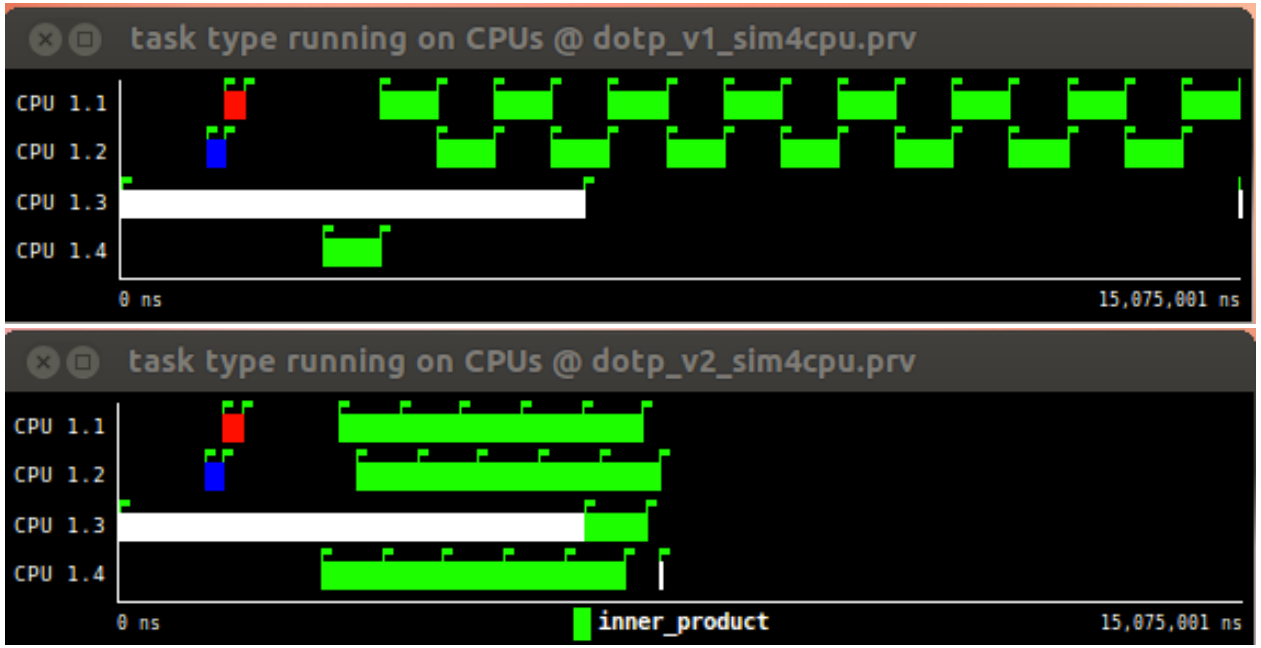
- CPU view load configuration



- Maximum concurrency view configuration



Dot product: Dimemas simulation comparison, N=16



- Comparison at same time scale

Outline

Motivation

API and example of use

Installation

Installation

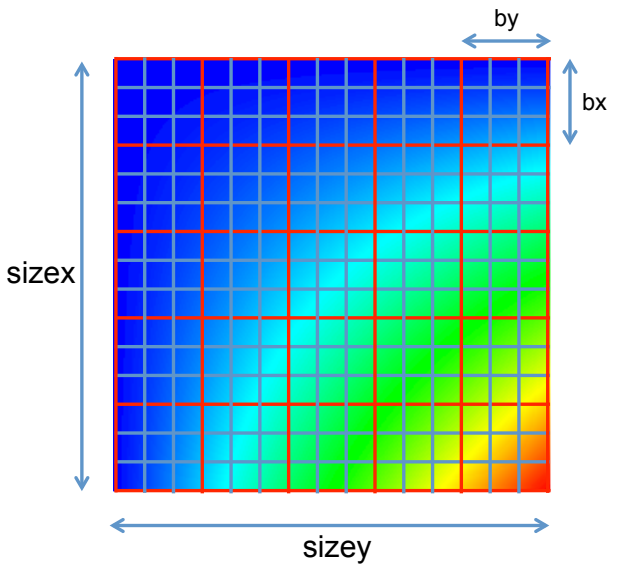
- Already installed in **Minotauro**, paths for library and include file set in `environment.bash`
- Installation of **tarball for Linux**, which can be downloaded from `http://bscgrid06.bsc.es/~tareador/download`
- Also available as a **Web portal** with execution on the Grid: `http://bscgrid06.bsc.es/~tareador`
  - Useful for interactive hands-on and demo sessions
  - Simple task graph and simulation visualizers
  - Timeouts in each step of the process
  - Some limitations (single file, no input files, ...)

Part II

Hands-on

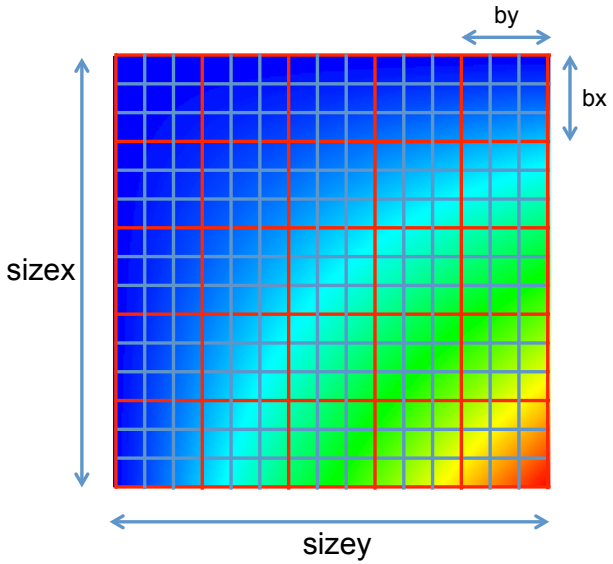
Solving the heat equation

```
iter = 0;
while(1) {
    switch( param.algorithm ) {
        case 0: // JACOBI
            residual = relax_jacobi(param.u, param.uhelp, np, np);
            // Copy uhelp into u
            for (i=0; i<np; i++)
                for (j=0; j<np; j++)
                   param.u[ i*np+j ] = param.uhelp[ i*np+j ];
            break;
        case 1: // GAUSS
            residual = relax_gauss(param.u, np, np);
            break;
        case 2: // RED-BLACK
            residual = relax_redblack(param.u, np, np);
            break;
    }
    iter++;
    // solution good enough ?
    if (residual < 0.00005) break;

    // max. iteration reached ? (no limit with maxiter=0)
    if (param.maxiter>0 && iter>=param.maxiter) break;
}
```



Solving the heat equation (Jacobi solver)

```
double relax_jacobi (double *u, double *utmp,
                     unsigned sizex, unsigned sizey) {

double diff, sum=0.0;
int nbx, bx, nby, by;

nbx = NB; bx = sizex/nbx;
nby = NB; by = sizey/nby;

for (int ii=0; ii<nbx; ii++)
   for (int jj=0; jj<nby; jj++)
      for (int i=1+ii*bx; i<=min((ii+1)*bx, sizex-2); i++)
         for (int j=1+jj*by; j<=min((jj+1)*by, sizey-2); j++) {
            utmp[i*sizey+j]= 0.25 * (u[ i*sizey     + (j-1) ]+
                                     u[ i*sizey     + (j+1) ]+
                                     u[ (i-1)*sizey + j     ]+
                                     u[ (i+1)*sizey + j     ]);
            diff = utmp[i*sizey+j] - u[i*sizey + j];
            sum += diff * diff;
         }
return sum;
}
```



Solving the heat equation (hands-on outline)

1. Try the initial task definition (Jacobi solver, one task per computation of block)
2. Which dependence is causing the serialization of all tasks?
3. Filter the object that causes the dependence (`tareador_disable_object` and `tareador_enable_object`)
4. Dimemas simulation with different number of processors
5. What else could be parallelized (or sequentially optimized)?
6. Repeat process with Red–Black solver
7. Repeat process with Gauss-Seidel solver

Exploring task decomposition strategies with Tareador

Eduard Ayguadé and Rosa M. Badia

Barcelona. November 2012