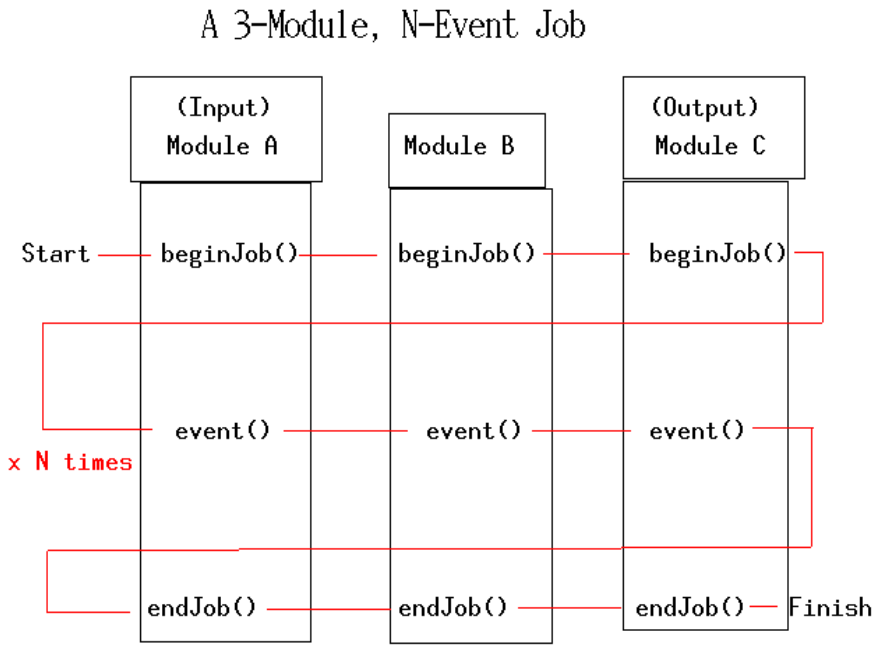
Parallelism at the module level:
scheduling and analysis

Introduction [1/2]

Data processing in the BaBar environment:

- Configuration done through Tcl files
- Analysis consists of a root object (path) containing sequences
- A sequence may contains other sequences or modules
- Module execution follows the insertion order (serial execution)



Introduction [2/2]

The SuperB Framework should be able to run more modules in parallel, but…

- How to define which modules can be executed at the same time?
- How to define an execution path?

Moreover, the analysis path has to be customizable at the user level, who may want to modify – to a certain extent – the «default» module sequences, or insert custom modules

- Scheduling has to be performed dinamically

Modules Definition [1/2]

Proposed solution: use module dependencies.

Each module has to declare which products it needs and which it produces.

Example:

```
Module    = RandomControl
Require   = EventList, EventID
Provides  = EngineList
```

Scheduling can then be performed dinamically

Modules Definition [2/2]

Specification of dependencies: just a note.

Requirements and products can be specified at compile time.

```
Class Module4 : public Module<module_four, Requires<product_one>,
Provides<product_three, product_four>
{
   bool operator()(Event& e) const {…};
}
```

(Thanks to F. Giacomini for the example)

Dependencies can also be specified at run time, losing compiler checks but allowing more flexibility.

Probably a mix of both is the right solution

Module Scheduling

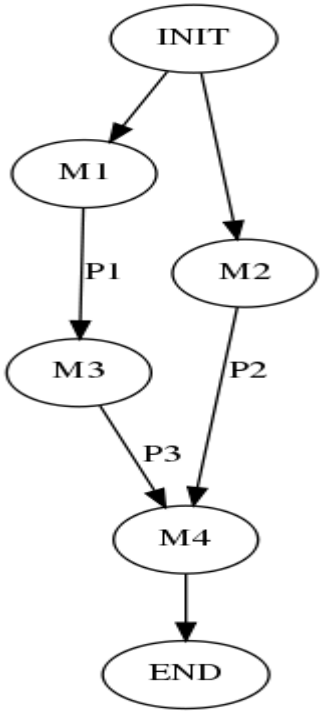
Using the exposed definition of a module, the Framework can produce a dependencies graph.

From dependencies graph it's easy to get the execution order.

The scheduler just has to:

- Checks for each module if the requirements are met
- In this case, schedules the execution of the module
- Adds products to the next level of the graph
- Repeats untill all modules are scheduled

```
Module = M1
Require =
Provide = P1
Module = M2
Require =
Provide = P2
Module = M3
Require = P1
Provide = P3
Module = M4
Require = P2 P3
Provide = P5
```



Scheduling Algorithm [1/3]

We have developed a working prototype of the scheduling algorithm that:

- Takes a generic list of modules with dependencies as input
- Produces a generic dependencies graph as output

A TBB graph can then be obtained from the dependencies graph.

Being a generic graph, the same algorithm can be used to instruct other schedulers (libdispatch?)

To check both performances and correctness, we have taken some measurements of the algorithm performances

Scheduling Algorithm [2/3]

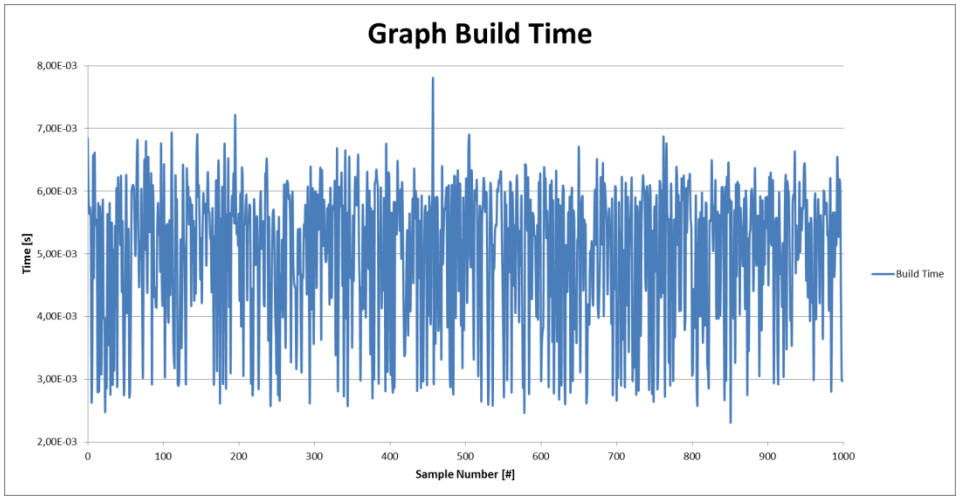
Benchmark setup :

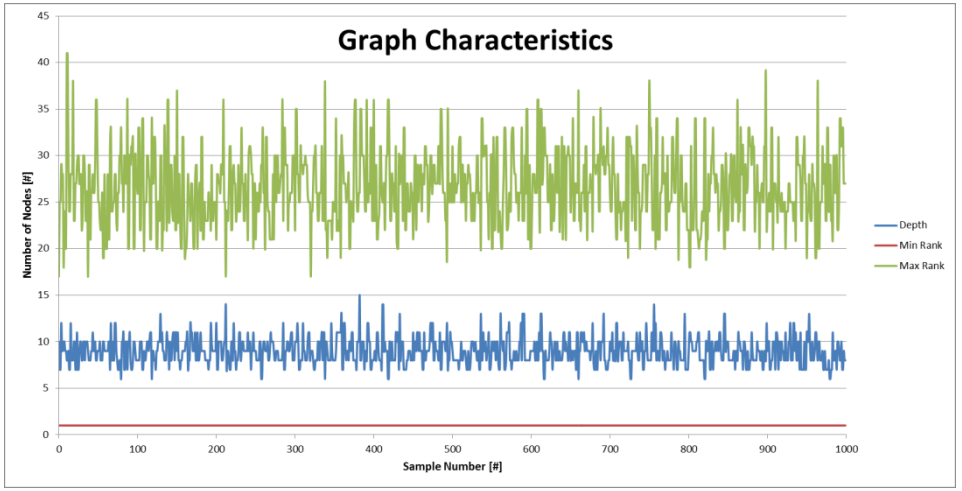
- 1k module lists
- 100 modules per list
- Random requirements (0-2) and products (0-4)

Population characteristics:

- Tree Depth = 6 ÷ 15
- Tree Rank = 1 ÷ 41

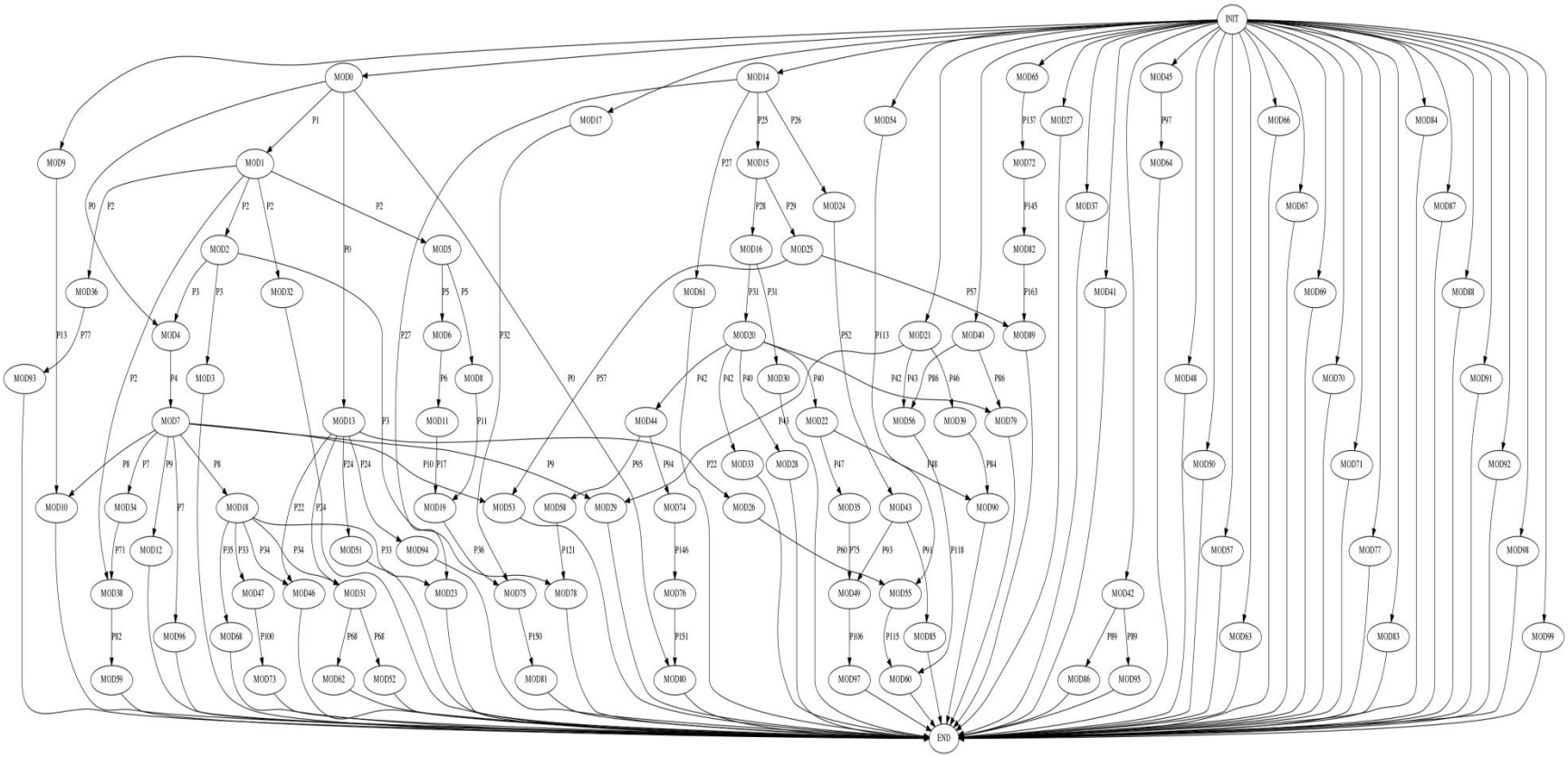
Execution time on an Intel Core i5-2400: less than 5ms





Scheduling Algorithm [3/3]

Here is how a produced dependencies graph looks like



Parallelism in FastSim [1/5]

We are investigating the parallelism level of FastSim, employing an analysis example (PacMC/snippet.tcl)

The analysis is configured employing tens of Tcl files

Each file may define sequences, enable modules, define parameters or include other Tcl files.

Extraction of sequences, modules and parameters must be done automatically.

A two stages approach is needed:

- Tcl file parsing to get modules and their parameters
- Source file parsing to get dependencies and defaults

Parallelism in FastSim [2/5]

Current Status:

- Recursively parsing of Tcl file: almost done.

We are able to get near all the modules in the analysis path, evaluate variables and get parameters set via «talkto».

- Parsing of associate module source files: still in progress

At present we are able to manage «standard» ::get and ::put instructions issued on the proxy, get parameters type and evaluate some variables.

Parallelism in FastSim [3/5]

A get example:

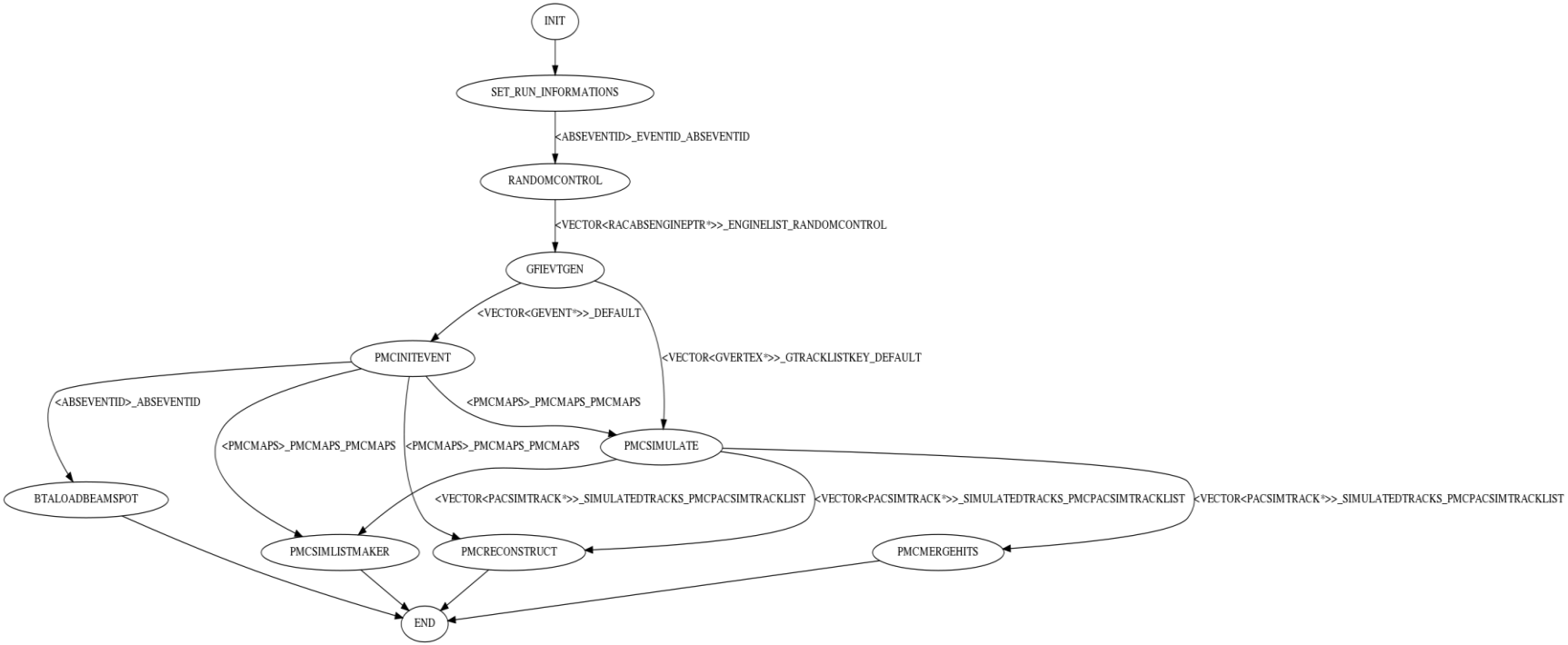
```
Ifd<<proxyType>>::get ( <AbsEvent>, <key> )
```

We are still working to solve some problems:

- Management of non standard get/put
- Variables scope
- OOP Polymorphism (i.e. get proxy type that is a son of the put type)
- Default values

Parallelism in FastSim [4/5]

Work is still in progress. Currently we can extract partial graphs of the analysis path like the following one



Parallelism in FastSim [5/5]

Some observations and questions:

- Even if the analysis is still partial, FastSim exposes some parallelism at the module level. We should try to exploit it

Regarding the module definition:

- Does the proposed definition of a module with requirements and products fit our needs?
- In FastSim some modules modify event properties (ex: BtaSelectCandBase). Is this a requirement we can avoid?
- Is there any module written being aware of its execution position in a sequence?

Thanks
For your attention!