Outline	Where we were last time	A (temporary) change of strategy	Where we are now	Conclusions
0	0000	000	0000000	00

Updates of R&D activities on SuperB analysis framework

Marco Corvo

CNRS and INFN

September 20th 2012

1 / 19



Outline	Where we were last time	A (temporary) change of strategy	Where we are now	Conclusions
•	0000	000	0000000	00

Outline

- Brief review of past activities
- A (temporary) change of strategy
- Where we are now
- Open issues
- Onclusions

- 一司

イロト イポト イヨト イヨト

A concurrent framework

(The first slides are grabbed from our presentation given at NYU for CHEP 12) $\,$

The current SuperB analysis frameworks share some features which prevent parallelism

- Written long time ago thus suffering from severe lack of modern programming paradigms
- Worst of all intrinsically serial

First step toward the parallelization of SuperB Framework is the analysis of current code, mostly based on BaBar legacy code, specifically one of the executables of Fast Simulation.

The analysis of a particular dataflow has a main goal:

• Factorization of the workflow

Outline 0	Where we were last time	A (temporary) change of strategy 000	Where we are now 00000000	Conclusions		
Measuring hidden parallelism						

The starting point was a specific Fast Simulation executable whose data flow includes 127 modules

- The analysis of the workflow has been performed
- For each module the analysis extracts:
 - The list of **required input** or **data products** needed by the module to run
 - The list of provided output generated by the module
 - The event processing time
- Basically the trick is to look inside the *Event* and dive into physics data products to understand who provides or requires what
- These lists are used to build an adjacency matrix, a means to represent the connections among vertices of a graph

(本語) (本語) (本語) (二語)



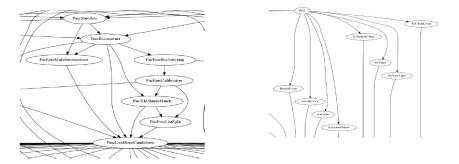
This analysis shows that the current code of Fast Simulation could benefit from modules parallelization.

- In particular, for this specific case, we found that the **tree** has ten levels, each with twelve nodes on average
- This clearly suggests that, on average, twelve modules could be scheduled in parallel

Num of modules	127
Graph Depth	10
Min Rank	1
Max Rank	54
Avg Rank	12



These are snapshots of the complexity we have to deal with. A big effort has been done to extract the dependencies of the modules, as the only source of information are the data products that modules write into the event, the data structure where physics results are stored



- (A 🖓



One of the results that made us confident to be able to improve dramatically the performances of our framework was the amount of time spent by this particular analysis into the "parallelizable" set of modules

- If the analysis were to spend 90% of the wall time into "parallelizable" modules, then exploiting concurrency would result in gaining a speed-up of (roughly) $S = \frac{T_{wall}}{\frac{T_{wall}}{T_{moduleY}}}$
- The timer we had (BaBar legacy code) had a bug
- This led us to change our approach to the problem, that is to try to increase the number of events analyzed (move from module level to event level parallelism)

イロト イポト イヨト イヨト

Outline Where we were last time 0 0000 A (temporary) change of strategy $\circ \bullet \circ$

Where we are now 00000000

Conclusions

Moving to event parallelism

- \bullet The first attempt was to exploit $\mathsf{Intel}^{\texttt{R}}$ Tbb <code>parallel_for</code>
 - This is the simplest approach as with minor changes in the main loop we are able to inject more than one event into the analysis chain
- The class parallel_for applies a so called "functor", the physics *module*, to a range of objects, the *events*

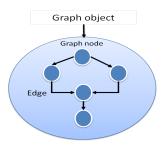
イロト 不得下 イヨト イヨト

Integrate with module parallelism

- The parallel_for approch allows only palallelism at the event level (we inject more that one event in the analysis chain)
- Our goal is to exploit also module level parallelism (allow modules to run concurrently on the same event)
- This is the approach that we want to pursue despite the initial bad performances
 - Because it is a valid and general model that can improve even more the performances
- This was done using Tbb flow::graph objects where every module in the analysis chain is mapped onto a graph node

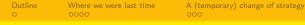
Outline	Where we were last time	A (temporary) change of strategy	Where we are now	Conclusio
0	0000	000	0000000	00

Module parallelism



Graph example

- To map the modules of the analysis chain to graph nodes, we analysed the event which is the container where all physics products are read and written
- The result is a list of requirements and products for each module that we use to build a graph of dependencies
- Major effort required to modify modules according to Tbb schema as modules must be implemented as functors



Features of this work

In this work we introduced some features

- Modules have been wrapped by a functor which calls the event method on the event
- Every module has been instrumented to extract two lists: one contains the requirements for the module to operate, the second contains the products of the module itself
- A method has been added to the class Framework to build a Tbb flow::graph object
- The event has been made "thread safe" substituting the list of physics products with a Tbb concurrent_hash_map
- A mechanism to limit the "injection" of events into the graph has been implemented

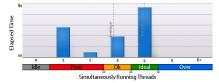
Outline 0	A (temporary) change of strategy	Where we are now	

Results I

2 0					/work/Framework/FastSir	n/parallel - Intel VTune
<u>F</u> ile View Help						
📓 🕼 🏠 🖆 📦 🕨	🗳 🕕					
Welcome rogglw 🗷						
🖉 Locks and Waits	- Locks and	Waits 😕 🧿)			
🗧 😌 Analysis Target 🛝	Analysis Type	3 Summary	ቆ Bottom-up	💑 Top-down Tree	🔁 Tasks	
motex oxoue/co/r	0.5265	1,423				
TBB Scheduler	0.2165	2				
[Others]	0.2555	2 / 20				

Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable states in the CS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.



📀 CPU Usage Histogram 🐚

This histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific number of CPUs were running simultaneously. CPU Usage may be higher than the thread concurrency if a thread is executing code on a CPU while it is logically...

Results with parallel_for (HT off)

イロト イポト イヨト イヨト

12 / 19

Marco Corvo (CNRS and INFN) Updates of R&D activities on SuperB analysis September 20th 2012

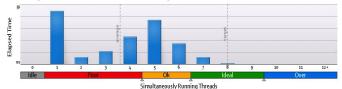
Outline O	A (temporary) change of strategy	Where we are now	Conclusions 00

Results II

20					/work/Fra	mework/FastSim/	/parallel - Intel VTune
<u>F</u> ile View Help							
📓 i 🌆 i 🏝 🖆 📦 i 🕨 🖻	; (D						
Welcome r100lw 🗷							
Locks and Waits - L	ocks and	Waits 🗶 🧿					
🕢 \varTheta Analysis Target 🔺 An	alysis Type	Collection Log	🕯 Summary	🗳 Bottom-up	🚯 Top-down Tree	🗄 Tasks	
NWE THE WORLD	0.75/5	14					
Stream oxdo411819	0.5145	359					
[Others]	0.9895	3,447					

📀 Thread Concurrency Histogram 🖻

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.



Results with parallel_for (HT on)

イロト イポト イヨト イヨト

13 / 19

Marco Corvo (CNRS and INFN) Updates of R&D activities on SuperB analysis September 20th 2012

Outline	Where we were last time	A (tempora
0	0000	000

A (temporary) change of strategy

Where we are now

◆ロト ◆聞ト ◆ヨト ◆ヨト

E 990

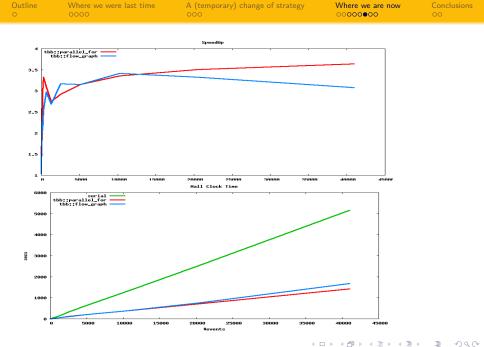
Conclusions

Results III

Die Voor Help ■ ① ① ② ③ ③ ③ ③ ③ ● ③ ④ ■ ② ③ ③ ③ ③ ● ③ ● ③ ● ③ ● ③ ● ③ ● ③ ● ③ ●	1				
Image: The set of the			Avork/Framework/FastSim/parallel - Intel VTune Amplifier		0 (
Nikiture Tegle Intel Varia A generation Nikiture Control Nikiture Nikiture<	EØ				
Locks and Waits - Locks - Lock	2	1 1 1 1 1	0) # 0		
Behavioria Tagel: Adalysis Tagel: Calcelona Log: Statumenty A Educationality	W	ekome rogg	# 8		
• (************************************	8	Locks and	Waits - Locks and Waits 🧶 📀		Intel VTune Amplifier XE 20
CPUIday		🖲 Analysis T	get 🛛 Analysis Type 🗮 Collection Log 🗵 Summary 👍 Bottom-up 🔥 Top-down Tree 📴 Tasks		
2010/24 2010/24		0/0+0-0+	αφι 10 15 15 15 15 15 15 15 15 15 15 15 15 15		Waiting Call Stack
TiteWork Provide <				Vaits Vaits	Viewing 4 1.0f2 ≥ selected st 87.6% (6.675 of 7.623)
Die Wohner Die Wohner <thdie th="" wohner<=""> Die Wohner Die Wohn</thdie>		Thread (ακ ₄		I Transitions	PadMCApp · PmcSimulate:e.
Iteration Iteration <t< td=""><td></td><td></td><td></td><td>👪 CPUTime</td><td>PadMCApp · AppSequence:e.</td></t<>				👪 CPUTime	PadMCApp · AppSequence:e.
CPUlog			a second data in the second		PadMCApp - Body:soperator(
CPUBage CPUBag	F				
CUUlay					
GPUloge Fall Kips Application Fall Kips Application Fall Kips Application					PadMCApp - operator new - t.
CPUllogr					
CPUUloge Patrixip-typertext. CPUUloge Patrixip-typertext. Tread (c. Patrixip-typertext. Patrixip-typertext. Patrixip-typertext. Patrixip-typertext. Patrixip-typertext.					
CVUlog			7		
PadAC/sp-fid/liasExente		CPUUsage			
Pad/Cop ItGMasterut		Thread Co			
the filters are applied. 🛊 Process Aus/Process 🔍 thread: AnyThread 🔍 Module: Anythodale 🔍 Unitatione: AnytOlicatione: 🔍 Call Stack Mode: Call Structure Functiones. 🔍 on file Mode: an 🔍					PadMCApp - ftdAliasExecute -
	×	No filters are	ppied. 😝 Process: Any Process 🗸 🏹 Thread: 🗸 Module: 🗸 Module: Any Madule 🗸 Utilization: Any Utilization: 🗸 Call Stack Mode: C	Inly user functions	✓ InlineMode on ✓

Concurrency breakdown (HT off)

Marco Corvo (CNRS and INFN) Updates of R&D activities on SuperB analysi: September 20th 2012 14 / 19



Marco Corvo (CNRS and INFN) Updat

Updates of R&D activities on SuperB analysis

September 20th 2012 15 / 19

Outline O	Where we were last time	A (temporary) change of strategy	Where we are now	Conclusions 00
Issues				

There's a intrinsic limit in Tbb flow::graph as regards the management of nodes

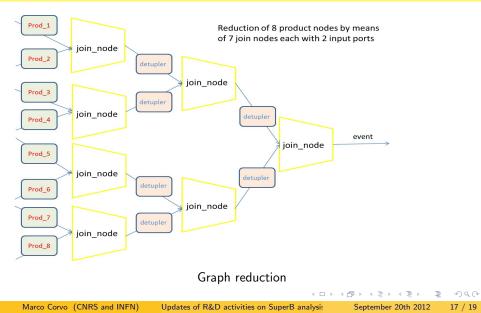
- The schema works so that a given module runs when all its "required" products are available
- If module A needs N products to run, we need to notify A when they are all available
- This is possible using a particular flow::graph node called join_node
 - This node has a mechanism which forwards a message to its successors only when all of its input ports have been filled

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ ● ● ●

- The issue with this node is that the number of input ports must be declared in advance and not dinamically
- The solution is to recursively reduce the graph combining join nodes in couples

oooo	000	000000	00
	A (temporary) change of strategy	Where we are now	Conclusions

Graph reduction



	Where we were last time	A (temporary) change of strategy 000	Where we are now	Conclusions		

- We encountered another problem on our way towards parallelism: a deadlock which prevents us from injecting more than 8 events into the graph
- This problem rises as we lock a particular module
- Locks are currently needed on certain modules to guarantee that only a given event can use it
 - This happens because modules are instantiated once at graph creation and we must avoid that two events access the same "common blocks" messing up the variables
 - Luckily this is true only for a limited number of modules, while for others we modified things to have thread safeness (e.g. data locality)
- The limit is equal to the number of threads that Tbb spawns to perform calculations
 - I. e. we knock against this deadlock when we inject 9 events
- Problem under investigation...

A

DD IIMIT (

• During our test we also saw what is likely to be a memory leak

Outline O	Where we were last time	A (temporary) change of strategy	Where we are now	Conclusions ●○
Short	and mid term	plans		

- We need to come to a full understanding of all dependencies among modules
- This is a good starting point, but there are a lot of analysis patterns to consider, not to mention a full reconstruction framework
- This exercise was helpful to understand also what modules can do and what modules must NOT do to improve optimization
- There's room for improvements also in the generation part, trying to limit (or at least factorize) the use of Common Blocks in Fortran code

Outline O	Where we were last time	A (temporary) change of strategy	Where we are now 00000000	Conclusions ○●
Conclu	usions			

- Our efforts so far have been rewarded as we shown the speedup gained by event and (partial) module level parallelism
- The (disclosed) parallel potential inside existing code is strategic
 - It optimizes resources usage and increases computing speed
 - Helps to better understand algorithms for future development
- Issues are still under investigation, but we are confident to be able to solve them
- In the long term we will abandon the current SuperB framework for a new one which is natively parallel and whose architecture will be designed based on our experiences