CMake versus CMT

Grigory Rybkin

Laboratoire de l'Accélérateur Linéaire
Orsay

working meeting, 9 November 2010

Plan

Plan

Plan

Plan

Working meeting CMT in ATLAS

- discussed CMake and CMT performance measurements made in Pere Mato's presentation, in particular, showed that for the GAUDI project, CMake was more rapid
  1. $\simeq$ 2 times with `make` simple
  2. $\simeq$ 4 times with `make -j8` said parallel
- it was needed to understand the origin of such differences
- for details see CMT in ATLAS, LAL, 7 October 2010

Code Optimisation

CMake configuration—in the tests being discussed—built code *without optimisation* while that of CMT—*with optimisation* `-O2`, expensive in terms of compilation time. This explains difference 1

Table: The measurements of elapsed time in seconds on a 16 core machine when I applied -02 optimisation for the GAUDI project

CMake +make+install	CMT/v1r22 cmt br make	CMake +make+install -j8	CMT/v1r22 cmt br make -j8
1080	1110	240	400

CMake still uses better `make` parallel

More Detailed Measurements

Table: The measurements of elapsed time in seconds on a 16 core machine for the GAUDI project with *make* simple

	CMake	CMT
generation	10	40
g++	960	1010
install	10	
total	1060	1150

More Parallelism with CMT

At the same time build independent packages in different

[tbroadcast] *threads*

[project level Makefile] *processes*

Table: The measurements of elapsed time in seconds on a 16 core machine for the GAUDI project

tbroadcast make	project level make	tbroadcast make -j8	project level make -j8
990	1120	360	350

Parallelism with CMake

At the same time build independent targets in different *processes*

Table: The measurements of elapsed time in seconds on a 16 core machine for the GAUDI project when I added to the CMake configuration dependencies between the packages equivalent to those of CMT

CMake with dep +make+install	project level make	CMake with dep +make+install -j8	project level make -j8
1080	1120	330	350

This explains difference 2

News from Pere Mato

Pere has

- managed to chain projects and export/import targets between them. This is important for simplifying the library dependencies
- moved all the Gaudi tests to CTest
- explored CPack to produce tar files or RPMs

Conclusion

- factor $\simeq 2$ of the CMake and CMT performances difference explained by the difference of code optimisations
- CMake may gain in performance thanks to building in parallel, depending on
  - the structure of the project and the packages
  - the availability of computing resources
- tbroadcast and project level Makefile give more parallelism with CMT
- explore how to introduce even more parallelism