A Virtual Geant4 Environment

Go Iwai
High Energy Accelerator Research Organization (KEK)
E-mail: go.iwai@kek.jp

Abstract. We describe the development of an environment for Geant4 consisting of an application and data that provide users with a more efficient way to access Geant4 applications without having to download and build the software locally. The environment is platform neutral and offers the users near-real time performance. In addition, the environment consists of data and Geant4 libraries built using low-level virtual machine (LLVM) tools which can produce bitcode that can be embedded in HTML and accessed via a browser. The bitcode is downloaded to the local machine via the browser and can then be configured by the user. This approach provides a way of minimising the risk of leaking potentially sensitive data used to construct the Geant4 model and application in the medical domain for treatment planning. We describe several applications that have used this approach and compare their performance with that of native applications. We also describe potential user communities that could benefit from this approach.

1. Introduction

Geant4 is a software toolkit for simulating the interaction of particles in matter. A Monte Carlo (MC) simulation using Geant4 takes into account the physics of particle interactions using theoretical models or experimental cross-section data. MC simulation can be of considerable help in the design of particle treatment facilities and the quality assurance of treatment plans.

Figure 1. Screenshots of PTsim

(a) Treatment port modelled in the PTsim for the Hyogo Ion Beam Medical Center, Japan. (b) gMocren visualises CT-scanned images and overlays particle trajectories and dose distribution in various views.

The Particle Therapy Simulation Framework (PTsim) was developed as a Geant4-based
simulation framework for the radiotherapy of cancer treatment with a special focus on proton and carbon therapy \cite{3}.

The PTsim contains a software application for modelling a treatment port consisting of a beam delivery system and a treatment head with patient data obtained from CT images. The PTsim provides a class library for geometry description, material definition, optimised physics process setting, scorers, event-level parallel processing, and a volume visualiser \textit{gMocren} which can overlay calculation results on the patient’s image as shown in Figure 1(b).

The PTsim already supports three Japanese proton and ion therapy facilities as well as three more in other countries. Figure 1(a) shows a PTsim example of a beam line for the Hyogo Beam Medical Center with simulated particles. The PTsim allows particle therapy clinicians or researchers to simulate their own facilities or envisaged facilities without requiring Geant4 expertise or programming skills.

The detailed simulation of the human body requires a considerable amount of CPU time to obtain sufficient statistical accuracy \cite{4}. In addition, the simulation has to be performed and the result needs to be obtained in the minimum time. Therefore, it is essential to perform the simulation in parallel with high-performance computing resources.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures.png}
\caption{PTsim job workflow on the PTsim Grid Platform and the Virtual Geant4 Environment}
\end{figure}

A PTsim-based common platform for MC dose calculation in Grid distributed computing systems has been developed to meet the above needs. Figure 2(a) shows a typical PTsim job workflow on the PTsim Grid Platform. This platform allows medical physicists to separate dose calculations (that are usually very large) into many small calculations which can then be processed in parallel over the distributed systems. As a result of this approach, significant performance improvement in turn-around-time for dose calculation has been achieved while medical physicists are inside the clinical facility \cite{5}.

There is, however, concern that a patient’s personal information might be obtained by unauthorised or unexpected users in the course of staging sensitive data, e.g. CT-scanned data, on the distributed systems.

2. Virtual Geant4 Environment
A Virtual Geant4 Environment (VirGen) consists of the application and data that provides users with a faster and easier way to access the Geant4 applications without having to download and build the software locally. Figure 2(b) shows a typical PTsim job workflow in the VirGen.
Medical physicists do not need to copy data to the external storage systems as shown in Figure 2(b). This approach results in a more secure dose calculation.

![Diagram](image)

**Figure 3.** The environment consists of data and Geant4 libraries built using the LLVM tools, which can then result in bitcode that can be embedded in HTML and accessed via the browser. The bitcode is downloaded to the local machine via the browser and can then be configured by the user.

Figure 3 illustrates the applications delivery sequence in the VirGen. To run applications with this environment, the Geant4 class library has to be compiled in advance by using the LLVM-based toolchain, which is a set consisting of a compiler, a linker, and other tools. Then applications are statically linked and translated into the bitcode executable, which is an OS- and architecture-independent format. The bitcode is distributed as an embedded object in HTML. When users open the HTML page and the applications are downloaded onto the local machine, the application code is verified, and translated into the hardware-specific code, which can run on Windows, Mac, Linux, and Google Chrome OS, with nearly 90% of native speed.

![Diagram](image)

**Figure 4.** LLVM – Low Level Virtual Machine is a compiler or compiler infrastructure within which to implement compilers.

An LLVM is a compiler or compiler infrastructure in which compilers can be implemented. A simple diagram illustrating the LLVM workflow is shown in Figure 4. An LLVM consists of...
a front-end and back-end. The front-end translates high-level programming languages, e.g. C, C++, Java, and JavaScript, into bitcode (LLVM-IR), which is a low-level programming language similar to assembly languages and is a language independent code. The back-end then translates the bitcode into the architecture- and hardware-specific code.

Native Client (NaCl) is a sandbox for efficiently and securely running compiled C and C++ code in the browser, independent of the user’s OS [6]. User applications compiled by LLVM-based compilers provided by NaCl can securely run in a web browser on multiple platforms. The secure execution of applications can be ensured by allowing the NaCl compilers to generate bitcode that differs from that of the original LLVM compilers. The bitcode can be verified by the Service Runtime of NaCl and contains restricted jmp and call instructions as well as the memory range that is accessible to the sandboxed applications [7].

NaCl provides a C++ standard library, the Pepper API, and the NaCl I/O library. Pepper provides a bridge to JavaScript and other browser resources. Pepper allows C/C++ applications to communicate with the hosting browser in a safe way. NaCl I/O is a utility library that provides implementations of standard C APIs such as POSIX I/O (stdio.h) and BSD sockets (sys/socket.h). Figure 5 shows the workflow of a basic NaCl application. Pepper allows applications to send and receive messages between C++ and JavaScript. C APIs provided by NaCl I/O, e.g. fopen, fwrite, fclose, allow C++ modules to access the HTML5 Filesystem that can be directly accessed from JavaScript.

Figure 5. NaCl (Native Client): Set of LLVM-based compilers and C/C++ libraries. Pepper: Communication between C++ and JavaScript. NaCl IO: POSIX I/O (fopen, fwrite, fread and fclose) and filesystems (memfs, html5fs, httpfs, and dev).
The minimal code snippets to handle the message “Hello CHEP” in C++ and HTML are listed in Listing 1 and Listing 2, respectively. The `send_message.cpp` (Listing 1) sends the message and the `nacl_embed.html` (Listing 2) receives and writes it into the browser console.

```cpp
Listing 1. send_message.cpp
pp::Instance::PostMessage("Hello CHEP");
```

```html
Listing 2. nacl_embed.html
<script type="text/javascript">
function handleMessage(message_event) {
    console.log(message_event.data);
}
</script>
<div id="listener">
    <script type="text/javascript">
        var listener = document.getElementById('listener');
        listener.addEventListener('message', handleMessage, true);
    </script>
</div>
<embed id="nacl_app_ID" src="nacl_app.nmf" type="application/x-pnacl">
```

3. Performance evaluation

As described in Section 2, NaCl generates the restricted bitcode that is sandboxed to run applications securely. This is one of the reasons for the performance overhead that is on average approximately 5% for SPEC2000 benchmark [8].

![Figure 6](image.png)

**Figure 6.** Benchmark result of Geant4 basic applications B1–B5 and PTsim.

Figure 6 and Table 1 present the benchmark result of the Geant4 basic applications B1–B5 and PTsim. The blue bars show the performance degradation of VirGen applications compared to native applications statically built for 32-bit architecture, and the red bars also
Table 1. Benchmark results. Times are CPU time in seconds.

|                  | PNaCl | Native | Performance degradation |
|------------------|-------|--------|-------------------------|
|                  |       |        | Slowdown to Native      |
|                  | 32-bit| 64-bit |
| CPU times        | static| dynamic| static                  |
| PNaCl            |       |        |                         |
| 32-bit           |       |        |                         |
| static           |       |        |                         |
| B1               | 41.46 | 34.76  | 30.88                   |
| B2               | 208.0 | 196.1  | 172.2                   |
| B3               | 40.00 | 38.30  | 38.06                   |
| B4               | 31.02 | 27.27  | 24.78                   |
| B5               | 138.3 | 112.2  | 103.3                   |
| PTsim            | 58.51 | 50.58  | 43.04                   |
| Slowdown to Native | 19%  | 6.0%   | 4.4%                    |
|                 | 34%   | 21%    | 5.1%                    |
|                 | 14%   | 25%    | 23%                     |
|                 | 34%   | 34%    | 16%                     |
|                 | 36%   |        |                         |

show the performance degradation of VirGen applications compared to native 64-bit applications dynamically linked to shared libraries. All applications in the VirGen are built with the NaCl compiler toolchain and static linking. These applications are 32-bit binary format and run as 32-bit executables due to the constraint imposed by NaCl. It is, therefore, a fair evaluation to compare the applications in the VirGen to native applications (32-bit/static). The performance degradation of the VirGen averages to 14% of that of static linking applications.

In this performance evaluation, a time built-in command for MacOS is used. All measurements are for Geant4 10.1.0 and Pepper 40 on a MacOS X (Yosemite/10.10) with a 2.8 GHz Intel Core i7 processor. The macro files, e.g. run1.mac and run2.mac, provided to the applications for measurements have been slightly modified from their original versions contained in Geant4. CPU times contain all the processes, i.e. geometry setup, material definition, creation of a cross-section table, and event production, except for static data file extraction.

4. Discussion
The VirGen undergoes performance degradation compared to the execution of regular applications. The benchmarks for basic Geant4 applications result in an average 14% performance degradation. Although the performance of different applications is likely to vary, this result should not have much of an impact for a wide set of user communities.

This benchmarking exercise only involved the evaluation of single-core processing. However, multi-core processing may still be improved. It should also be noted that the performance of applications that frequently access files on the HTML5 Filesystem and communicate messages is worse.

Based on the reasons stated above, the advantages of applying this technology to existing/non-existing applications are:

- Productive for communities in which developers and end-users are strictly separated due to zero deployment time.
- Beneficial for the communities in which end-users run applications securely without data exposure to the Internet.
- Reproducible for communities in which many end-users are widely distributed and where the results need to be verified with high accuracy.
- Portable for communities that roll out platform-independent applications.
- Fast implementation for communities that plan data delivery via cloud storage.
5. Conclusion and future works
This paper described the development of the Virtual Geant4 Environment (VirGen) in which applications were successfully run in a web browser on multiple platforms at near native speed, which was almost 90% of that of regular Geant4 applications. The PTsim application in the VirGen ran without requiring sensitive data to be uploaded to distributed systems and enabled the software to be built on the local machine.

As discussed in Section 4, there are not only medical domains but also potential user communities that could benefit from this approach.

In conclusion, the ability of the VirGen to minimise the risk of leaking data without any performance degradation has been practically proven in the medical domain as well as for several user communities.

In future, support for multi-thread processing and an OpenGL visualisation in the web browser is planned for the VirGen.

Acknowledgments
The author thanks A. Hasan, Centre for e-Research, King’s College London, for advice on English expressions.

References
[1] Allison J et al. 2006 IEEE Trans. Nucl. Sci. 53 270–278
[2] Agostinelli S et al. 2003 Nucl. Instrum. Meth. A506 250–303
[3] Kameoka S, Amako K, Iwai G, Murakami K, Sasaki T, Toshito T, Yamashita T, Aso T, Kimura A, Kanai T, Kanematsu N, Komori M, Takei Y, Yonai S, Tashiro M, Koikegami H, Tomita H and Koi T 2008 Radiological Physics and Technology 1 183–187
[4] Akagi T, Aso T, Faddegon B, Kimura A, Matsufuji N, Nishio T, Omachi C, Paganetti H, Perl J, Sasaki T, Sawkey D, Schumann J, Shin J, Toshito T, Yamashita T and Yoshida H 2011 Progress in Nuclear Science and Technology 2 912–917
[5] Aso T, Takase W, Iwai G, Sasaki T, Watase Y, Maeda Y, Yamashita T, Akagi T, Harada S, Nishio T, Cai S Y, Chao T C, Yen E, Wu T H and Lin Y T 2014 The International Symposium on Grids and Clouds (ISGC) 2014 PoS(ISGC2014)002
[6] Native Client http://gonacl.com/
[7] Sehr D, Muth R, Bille C L, Kihmenko V, Pasko E, Yee B, Schimpf K and Chen B 2010 19th USENIX Security Symposium pp 1–11
[8] Yee B, Sehr D, Dardy G, Chen J, Muth R, Ormandy T, Okasaka S, Narula N and Fullagar N 2009 30th IEEE Symposium on Security and Privacy pp 79–93