Aspect-Oriented Parallel Programming Using AspectC++ and OpenCL

Robert Clucas
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg, South Africa
robert.clucas@students.wits.ac.za

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

CCS Concepts

•Computing methodologies → Parallel programming languages; •Computer systems organization → *Parallel architectures;* **•Software and its engineering** → *Context specific languages;*

Keywords

Aspect; device; host; pointcut; advice

1. INTRODUCTION

In recent years parallel systems become more established as processor core frequencies began to plateau. The result of this was multicore systems such as CPU's with numerous cores and GPU's with many cores, as well as hybrid systems which make use of both numerous core CPU's and many core GPU's. The cores used for CPU's and GPU's differ in complexity. GPU's use many (up to thousands), simple cores to increase computational efficiency. CPU's use fewer (1 - 20), complex cores that allow for instruction level parallelism, however, as this complexity comes from the vast number of transistors that make up the processor which leads to a large amount of power being required by the processor [4]. General-Purpose GPU (GPGPU) programming involves combining CPU's and GPU's into a single, hybrid system which provides increased computational performance by using the CPU cores to pass data to the GPU cores which perform the computation in a parallel manner. MAYBE INCLUDE FIGURE.

For GPGPU programming there are two options available to the programmer, namely OpenCL [2] and CUDA [6].

OpenCL is an attempt to provide a standard API for programming parallel capable hardware. It provides support for all main hardware vendors, namely Nvidia, AMD and Intel. This wide range of support is advantageous in the sense that a parallel implementation using the OpenCL API could run on both the CPU and GPU cores present in the hybrid system, simultaneously. CUDA applies a similar methodology and also provides and API for writing programs that will run on parallel capable hardware, however, it is specific to Nvidia hardware and hence parallel kernels cannot be run on CPU's or GPU's from any other hardware vendor.

Admittedly, the latest version of the CUDA SDK does provide more advanced functionality as it supports C++11 features whereas OpenCL is based on C, but does provider C++ bindings. Due to CUDA only supporting Nvidia hardware, and the growing acceptance of OpenCL as a standard API, OpenCL was chosen for the implementation.

While these hybrid parallel systems follow Moore's law, they are more difficult to program when compared to traditional, sequential systems. There are numerous factors that cause the increase in difficultly. The main cause is the two different hardware types in the hybrid system, the CPU (from here on referred to as the *host* as it is termed in the GPGPU programming model) and the GPU (from here on referred to as the *device* as it is termed in the GPGPU model). This requires, firstly, the setup of the parallel context which enables paralllel kernels to be executed on the device but from the host. Secondly, memory transfer between the host and device as the computational data must be moved from the device memory to the host memory before computation and then from device memory back to host memory after the computation, Thirdly, while this is not strictly necessary for performing implementing a par-

SAICSIT '15 September 28–30, 2015, Stellenbosch, ZAR

DOI: 10.1145/1235

Figure 1: Process of weaving aspect and core code.

allel program it is, however, advantageous - knowledge of the memory structure of the device. GPU's often have complex memory heirachy's which provide memory as varying distances from the computational cores to increase computational performance.

The need for GPGPU programming arises from the computational complexity of the algorithm needing to be solved. The above mentioned problems are not related to the core computation and thus introduce overhead. The overhead is distracting and crosses the problem domain, 'tangling' the resultant code. Furthermore, these cross-cutting code is required in multiple places when the parallel implementation of the algorithm requires numerous kernels. Aspect-oriented programming (AOP) [3] can provide a solution to this which allows the cross-cutting components to be *woven* into the core computational code, resulting in code comprising only of components directly related to the core computation.

The AOP paradigm allows code components which do not lend themselves to an object-oriented solution to be modularized using aspects. The code components which have been modularized into aspects are then 'woven' into the core component code through the aspect compiler. This process allows the programmer to separate the core component code, which is written using any normal programming language, and the aspect code, which is written in the aspect language. Before being woven, the core component code consists solely of code related specifically to the problem - the algorithm in the case of parallel programming - and the aspect code contains all the elements that 'tangle' the core component code when present. After being woven, the cross-cutting aspects are combined with the core code to produce the fully functional application. Figure 1 shows the process of weaving the aspect and core component code to produce the final application.

This paper describes the use of aspects, using AspectC++, to hide the above mentioned complexities present in parallel programming from the programmer. The resultant programming model aims to allow for an environment that allows the implementation of parallel programs to be comparatively simple with a sequential, C++ only implementation, but that provides performance that is comparative to a non-aspect parallel implementation.

The implemented aspects deal with: Firstly, the setup of the OpenCL context that allows the host to execute the parallel kernels on the device, this included the management of the potentially numerous devices in the system. Secondly, The allocation and deallocation of memory on the device when kernels are executed, and thirdly, the execution of kernels when the host code requires the parallel computation to be executed.

The implemented aspects do not deal with the kernel itself and the type of memory that used on the device, i.e the aspects do not determine if the kernel should use global or shared memory. This choice was made as it was identified that the kernel directly related to the problem domain and the freedom to implement the kernel and define the grid size for the kernel execution should be given to the programmer.

The solution allows the programmer to not have to learn the complex CUDA ot OpenCL API's as the aspects hide these complexities from the programmer. However, as the kernels are the responsibility of the programmer, the programmer will require an understanding of how parallel hardware executes a kernel and how the indexes of the kernel grid relate to the computation.

The rest of the report is structured as follows: Section 2 reviews work related to the use of aspects in a parallel context; Section 3 describes the use of aspects with AspectC++, and its features to achieve the above mentioned goals; Section 4 presents the results of the implemented solution to two problem domains; Section 5 provides a comparison of the parallel aspect implementation with that of a non-aspect non-parallel C++ solution, and a non-aspect parallel OpenCL solution, in terms of both performance and code complexity as well as a comment on the limitations of the parallel aspect solution; Section 6 concludes and Section **??** discusses directions of exploration for future work in this area.

2. RELATED WORK

3. ASPECTC++ IN PARALLEL PROGRAMMING

4. RESULTS

5. EVALUATION AND LIMITATIONS

6. CONCLUSION

7. FUTURE WORK

The *proceedings* are the records of a conference. ACM seeks to give these conference by-products a uniform, high-quality appearance. To do this, ACM has some rigid requirements for the format of the proceedings documents: there is a specified format (balanced double columns), a specified set of fonts (Arial or Helvetica and Times Roman) in certain specified sizes (for instance, 9 point for body copy), a specified live area (18 × 23.5 cm [7" × 9.25"]) centered on the page, specified size of margins (1.9 cm [0.75"]) top, (2.54 cm [1"]) bottom and (1.9 cm [.75"]) left and right; specified column width (8.45 cm [3.33"]) and gutter size (.83 cm [.33"]).

The good news is, with only a handful of manual settings[1], the LaTeX document class file handles all of this for you.

The remainder of this document is concerned with showing, in the context of an "actual" document, the LaTeX commands specifically available for denoting the structure of a proceedings paper, rather than with giving rigorous descriptions or explanations of such commands.

8. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller

[1]Two of these, the `\numberofauthors` and `\alignauthor` commands, you have already used; another, `\balancecolumns`, will be used in your very last run of LaTeX to ensure balanced column heights on the last page.

sections. The command `\section` that precedes this paragraph is part of such a hierarchy.[2] LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

8.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[3] typeface, but that is handled by the document class file. Take care with the use of[4] the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[5].

8.2 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

8.2.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[5]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

8.2.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context.

[2]This is the second footnote. It starts a series of three footnotes that add nothing informational, but just give an idea of how footnotes work and look. It is a wordy one, just so you see how a longish one plays out.

[3]A third footnote, here. Let's make this a rather short one to see how it looks.

[4]A fourth, and last, footnote.

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

8.3 Citations

Citations to articles [**?**, 1, **?**, **?**], conference proceedings [1] or books [**?**, 5] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [5]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[5].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed or supported.

8.4 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide.*

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a

Figure 2: A sample black and white graphic.

Figure 3: A sample black and white graphic that has been resized with the `includegraphics` command.

single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

8.5 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** files to be displayable with LaTeX. If you work with pdfLaTeX, use files in the **.pdf** format. Note that most modern TeX system will convert **.eps** to **.pdf** for you on the fly. More details on each of these is found in the *t*he fully functional application. Figure 1 shows the process of weaving the aspect and core component code to produce the final application.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

8.6 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

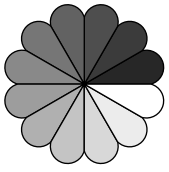

Figure 5: A sample black and white graphic that has been resized with the `includegraphics` command.

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

9. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

10. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you.

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

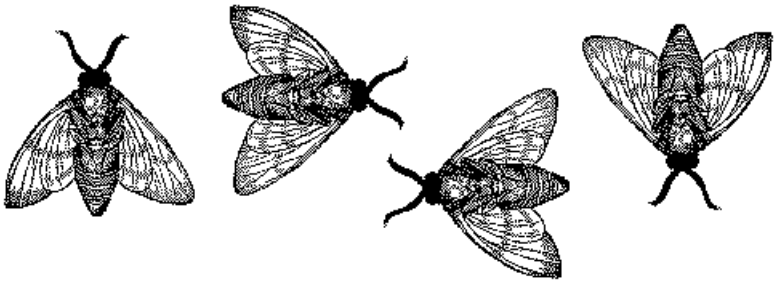

Figure 4: A sample black and white graphic that needs to span two columns of text.

In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

11. REFERENCES

[1] M. Clark. Post congress tristesse. In *TeX90 Conference Proceedings*, pages 84–89. TeX Users Group, March 1991.

[2] Khronos Group. *OpenCL 1.0 Specification*, Dec. 2008. Rev. 29. https://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. *European Conference on Object-Oriented Programming*, Springer-Verlag LNCS(1241):220–242, June 1997.

[4] R. Kumar, K. Farkar, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA hetrogeneous multi-core architectures: the potential for processor power reduction. *Annual IEEE/ACM International Symposium*, MICRO-36:81–92, 2003.

[5] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[6] Nvidia. *CUDA RUNTIME API*, March. 2015. http://docs.nvidia.com/cuda/index.html#axzz3cGMEdjIx.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\addition-alauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately

experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.