A CASE FOR SCIENTIFIC APPLICATIONS ON SMART PHONES

V. Ganesh[1]

[1]*V. Ganesh, Post Doctoral Fellow,*
Department of Computer Science,
Australian National University, Australia
E-mail: ganesh@cs.anu.edu.au

Abstract

Processors used in mobile phones are increasingly becoming powerful and capable of handling much of what a desktop counterpart would do, albeit with much lower power requirements. With the ubiquity of these devices, and increasing availability of unused processor cycles, it is worth while to investigate the kind of applications that can tap into this resource. In view of this, compute intensive scientific applications are suggested as a possible candidate. This has two-pronged benefits; one developing scientific applications on power efficient and ubiquitous processors would bring in new ways to speed up applications, not necessarily conceivable using current desktop processors. This scenario would also include the possibility of developing large scale distributed applications, which might be beneficial merely due the ubiquity and inexpensive nature of the target device. Secondly, a subtle, but equally important benefit is the use of such applications in many scenarios where a desktop or a laptop PC is simply too cumbersome to use. Such scenarios would include practical demonstration of concepts in classroom teaching or even as a marketing aid.

Keywords: mobile processors, scientific applications, mobile applications

1. INTRODUCTION

Not so long ago, Kozyrakis and Patterson [1], envisaged a new direction towards development of computer architecture specifically targeted towards mobile devices. More than a decade later, the continuing growth of telecommunication industry and mobile networks have seen considerable interest in developing all-encompassing personal communication devices, generally termed as smart phones. The core of these devices typically constitutes of two processors: a communication- and an application-centric processor. The application centric processor is capable of running a fully capable multitasking OS [2] such as Symbian, Android or Windows Phone OS. The current article will focus on the ability of application processor on smart phones to handle compute intensive scientific codes.

1.1 Ever Powerful Mobile Processors

Competing with their power hungry PC class microprocessors, smart phone processors, primarily designed by ARM [3], are beginning to touch 1GHz clock speeds, albeit consuming a fraction of power. Generally speaking, higher clock ratings do not necessarily mean more compute capability, but broadly speaking a similar design of a processor with higher clock rates will perform faster. What is interesting in smart phone processor market is that though the top level processor speeds have greatly increased from below 100MHz only a few years ago to about 1GHz currently, there has been considerably less impact on power consumption of these processors. Owing to these low power features, applications developed to take optimum advantage of them would require to have differing characteristics in comparison to desktop counterparts. Further, as these processors are low power (typically consuming less than 0.5 watts), devices that use these can be conveniently powered by portable rechargeable batteries. This scenario opens up whole range of possibilities such as using these devices as a demonstration tool, for instance in a science class or quick small scale demo of a full-fledged application by a sales person eager to sell a new product.

As a typical smart phone, Nokia E51 device [4] is used for the current study. This is equipped with an ARM 11 processor core clocked at 369MHz and has a total of 96 MB of RAM. The processor is equipped with 8-stage integer pipeline with separate load-store and arithmetic pipelines. It also supports branch predictions and return stack. An interesting aspect, particularly for scientific applications is the ability to attach an optional hardware based support for floating point operations. Devices with such capabilities, however, do not seem to be available widely in the current mobile device market.

Invariably, when considering to port compute intensive applications on mobile devices one has to take into account the following issues:

1. Generally lack of support for floating point hardware. This would eventually hit performance for real applications.

Though, this aspect might improve in future revisions of mobile devices, it is nevertheless important to consider.

2. Available memory and secondary storage on mobile devices would pose serious limitations to the nature of jobs that can be handled with ease.

3. Power consumption of the methods ported on mobile devices is also of considerable importance, as this would mainly affect the usability of methods on a devices powered primarily by battery source.

This article tries to highlight the above issues, along with other subtle but important ones, by porting a rather complicated scientific application to the above mentioned mobile device. Along side, an investigation of its usability on a smart phone is presented. At this point, it should be noted that though smart phones have been used in variety of ways by scientific community lately, to the best of author's knowledge there has been no attempt to look into feasibility of putting in computationally extensive codes on these platforms.

1.1 Quantum Chemistry

Modern day microprocessor technology has been ultimately driven by some of the highly demanding compute intensive scientific (and lately financial and multi-media) applications. Quantum chemistry is one such early application that has a heavy user base as well as a sizable developer base (both in terms of theory and method development). Most of quantum chemistry essentially revolves around finding an accurate enough solution to the Schrödinger equation [5] for molecular systems:

$$\hat{H}\psi = \epsilon\psi \quad \ldots (1)$$

where, ψ is the molecular wavefunction, $\hat{H}$ is Hamiltonian operator and ϵ is the energy of the system. Although Schrödinger equation is not fundamentally solvable for multi-electron systems, approximations given by Hartree and Fock [5], or in short HF, is widely used by quantum chemists. In the ensuing sections, an implementation of complete Hatree-Fock algorithm for a smart phone and a few important issues encountered in the process are reported.

2. QUANTUM CHEMISTRY CODE ON MOBILE

Due to wide range of available mobile platforms it is difficult to decide on the most suitable platform for experimenting and testing. For the sake of convenience, S60 platform [6], promoted by Nokia is chosen. Another reason for this choice is driven by the ability to natively compile codes on the phone with S60 platform running Symbian OS. Further, the S60 platform supports scripting via Python [7], allowing one to rapidly develop skeletal code and then optimize the compute intensive codes with native C/C++ implementation. For the purpose of this study, an open source code developed by the author for performing HF energy calculation on S60 phones, termed mobihf [8] is used. This code is a full featured S60 application (described later in the article, Section 2.3), the core of which is a compute engine that solves the following pseudo-eigen value problem:

$$FC = \epsilon SC \quad \ldots (2)$$

which is an approximate solution to the Schrödinger equation. In the above equation, F is the Fock matrix, C is the coefficient matrix and S is the overlap matrix, while ϵ are the eigenvalues representing the orbital energies of the system. All the matrices are of dimension $N \times N$, for a molecular system represented with N basis functions. A solution to this equation is achieved using an iterative self consistent field (SCF) procedure. The most expensive part in the above equation is setting up of the Fock matrix (F) which requires one-electron H^{core} matrix, density matrix P and the two-electron integrals $\langle ij|kl\rangle$:

$$F_{ij} = H^{core} + \sum_{k,l}^{N} P\left[\langle ij|kj\rangle - \frac{1}{2}\langle ik|lj\rangle\right] \quad \ldots (3)$$

Of which the four centered two-electron integrals are some form of Gaussian functions representing electron-electron repulsion in a molecular system. Though the chemistry involved in the above equations is quite intricate (See Ref. [5], for those interested), what is important to understand here is the inherent compute intensive nature of this method. Note that for a system represented using N basis functions, there would typically be $O(N^4)$ two-electron integrals to compute. In essence, this forms the largest bottleneck in implementing an HF code on any processor.

The mobihf computational kernel is coded with a mix of object-oriented and procedural approach. While classes are used to represent integrals, basis functions and SCF procedure, many pure functions are coded in C to actually implement methods of Integral class. The graphics part of mobihf uses the simple graphics support provided via PyS60. One can also use OpenGL for prettier presentation, however these are not implemented at the moment. The computational kernel heavily relies on mathematical routines provided by the standard C library, some of these implementations on S60 platform, however, have problems that could be crucial. The following section examines a few of these issues.

2.1 Issues With Math Library

Owing to the knowledge of compute intensive portions of HF method, the code is mostly written in Python with the code to evaluate two-electron integrals written as a native C library for improved performance. It is to be noted here that all the calculations are done with double precision types. However, since the test hardware does not have any in-silicon floating point units, an emulation library provided

by the Symbian SDK [9] is used. This runtime has some quirks though, the most nasty ones are in math library that implement `acos()` and `pow()` functions that seem to crash the code completely for very small values of their arguments. To circumvent this problem a code similar to the following is used:

```
double mpow(double x, double y) {
  if (fabs(y)<=1.0e-10) return 1.0;
  if (fabs(x)<=1.0e-10) return 0.0;
  return pow(x, y);
}
```

Taking cue from this issue, one must be careful when porting a math intensive code to the smart phone platform as the libraries provided with these platforms may not be as thoroughly tested as is the case on the desktop counterpart.

2.2 Power Aware Programing

The HF procedure described above can be coded in two ways: conventional and direct. These basically refer to how the two electron integrals are evaluated and used. One way is to calculate them at the beginning of the SCF cycle, store at some location (usually secondary storage) and then utilize them for each SCF cycle. Such a procedure is called conventional SCF. Evidently, available storage is a limiting factor of how big a system can be handled if a conventional procedure were followed. Further, the retrieval of these integrals from secondary storage would become a bottleneck with increase in number of stored values.

Another method that is used routinely in modern day popular quantum chemistry codes is the direct method, where in the requisite two-electron integrals are computed on-the-fly. This method requires less storage, however the expensive two-electron integrals would need to be re-computed at each SCF cycle (with some of these being possibly re-computed with in a cycle). The rational behind using direct SCF procedure evidently is that, it is faster to just re-compute the integrals, than store and retrieve from a slower secondary storage. Apart from this factor, there is an obvious advantage of the ability of the code to handle larger systems.

For an implementation on smart phones powered by a battery source though, it is required to make a different consideration altogether: power consumption. A direct SCF approach would miserably guzzle up battery power. A more practical approach would be to use in-core conventional SCF approach. This approach, though practical has a limitation on the size of systems that could be handled, which is largely controlled by available main memory.

The current choice of using in-core conventional SCF however can be treated as opportunistic rather than a well thought out idea with the explicit intent of reducing power consumption. However, it is important to appreciate here that when porting compute intensive scientific applications from a PC environment to mobile platforms, one should thoroughly consider the power requirements of the software methods used. The fastest method on a PC environment need not necessarily be most power efficient for a mobile environment. Successive sections give a walk through of the mobihf application and take a look at its feasibility and possible usage scenarios of such scientific applications on smart phones.

2.3 mobihf: A Walk Through

Apart from implementing the core compute intensive HF method, mobihf also provides complete interface for reading (or modifying) input data in terms of coordinates of atoms describing a molecular structure. The interface also provides a basic viewer to display the 3D structure of molecules using a line and circle model (cf. Fig. 1). There are options included that allow one to specify the calculation parameters. These includes maximum SCF cycles that may be taken for a calculation and an energy cutoff used to indicate convergence criterion for the SCF procedure.

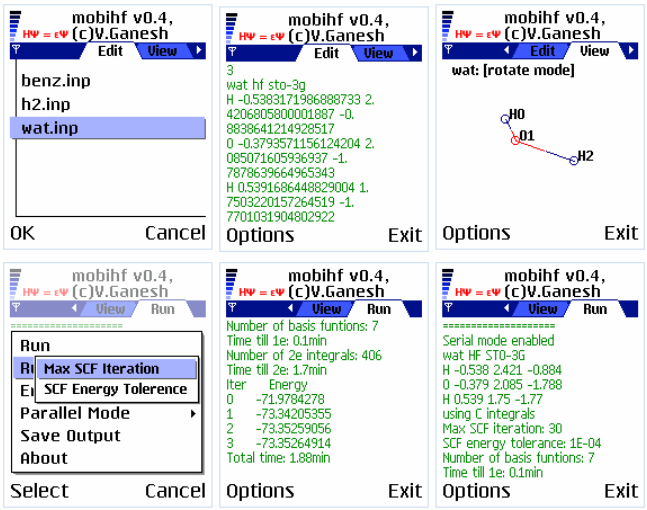

Fig. 1: A walk through of the mobihf application (running on Nokia 6600) which allows one to do a electronic structure calculations on small molecules. The interface provides options to read or modify input, visualize it and finally set up a calculation and view the results.

Using the interface itself one can submit the required calculation and view the final results (cf. Fig. 1). An important point to note here is that most of the mobile devices are constrained enough to have any useful command line interface. This issue is of practical importance given the fact that most of the scientific codes on the desktops predominantly rely only on command line, with only a few providing a GUI. However, on smart phones the only viable user interface is to use a GUI shell, which is a subtle but important difference when porting compute intensive standalone codes. The following section looks at the timing and other issues of using this application to perform few real calculations on small molecular systems.

2.3 Timings and Other Issues

To investigate the feasibility of actually running a calculation, a few of the test cases, albeit very small molecules are used. The results are summarized in Table 1. All the calculations are performed with STO-3G basis set

[10], which decides the number of basis functions (N_{bf} column in Table 1) used to represent the molecular system. This in-turn decides the number of two electron integrals in the system (marked as N_{2e} in Table 1).

For the purpose of reporting realistic time, it is to be noted that each calculation was run twice on full battery power and the least of the timing is reported here. Also, it was ensured that no background processes (apart from the necessary ones) were running during a calculation. This however has a few caveats, most notably the power saver option on the test device does not have an option of setting the time-off on screen saver for more than 1 minute. This means, timings reported for systems that take more than a minute might be slightly skewed, though this does not necessarily effect the general observations presented here.

Table 1: Timings (in minutes) for evaluating two electron integrals and SCF for a series of small molecules using conventional procedure. See text for details.

Molecule	N_{bf}	N_{2e}	T_{2e}	Total time
H_2	2	6	0.01	0.02
H_2O	7	406	0.28	0.32
NH_3	8	666	0.43	0.51
CH_4	9	1035	0.63	0.73
N_2	10	1540	1.42	1.52
O_2	10	1540	1.43	1.53
C_2H_4	14	5565	4.05	4.50
O_3	15	7260	7.17	7.72
CO_2	15	7260	7.50	8.10
C_6H_6	36	222111	195.97	214.93

As can be inferred from previous sections, the number of compute intensive two-electron integrals scales non-linearly and hence the timings are also affected in the same way. The timings for computing two-electron integrals are separately reported (cf. column T_{2e} in Table 1) along with total time spent for the calculation. Evidently, around 90% of the time is spent in computing two-electron integrals, and hence the rational for moving the code that computes two-electron integrals to native C. For small di- and tri- atomic molecules the timings are relatively low. For heavier di- and poly- atomic molecules the compute time increases exponentially. To stress the limit of the device in performing such a calculation, a test on benzene molecule was performed (C_6H_6 in the Table 1), which took a staggering 215 minutes to complete the job. For the sake of comparison, the same job run on a single core of modern Intel CPU @ 2.4 GHz with re-compiled mobihf took merely 46 seconds! Further, the test case run using a popular quantum chemistry suite [11], [12] took only 5 seconds on the above mentioned Intel hardware. However, these comparisons are quite unfair, given that mobihf is majorly written in Python with only some compute intensive parts written in C, along with other differences in algorithmic details of the code and more importantly, major differences in hardware capabilities (lower clock speeds and lack of hardware floating point unit, for instance). This, however, clearly suggests that such a device would not be practical to run some heavy calculation, nevertheless can well serve as demonstration of a concept.

It is quite noticeable to observe that during computation of the two-electron integrals, the power usage peaks (cf. Fig. 2), and then drops slightly when the SCF procedure starts. Given this fact, it is probably not a wise idea to re-compute the two-electron integrals at all, and in-turn, supports the decision to use in-core conventional SCF for the current study.

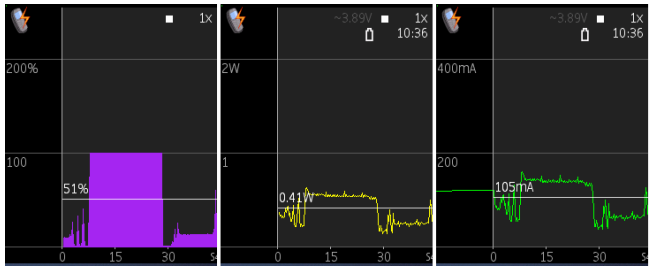

Fig. 2: CPU usage along with power usage profile for the system H_2O obtained using Nokia energy profiling tool. See text for explanation.

Note that even though, there would be a practical time bound on performing these calculations, theoretically it should be possible to run even larger systems. This argument is supported by the fact that the memory requirement for calculations of benzene molecule is little below 2 MB, where as the test device has a total of 96 MB. For such cases a collection of mobile devices with work distribution could be used. Applications that would scale over a large distributed network with minimal dependence on inter-device communication (HF code being a typical example) would benefit immensely in such a scenario. Though interesting, investigating practicality of such an approach is beyond the scope of this article.

3. LOOKING BEYOND NOVELTY

Development of a complex and computationally intensive quantum chemistry code could be considered a novelty in that it demonstrates the possibility and viability to build compute intensive application on smart phones. Moreover, these applications, though limited by compute resources, are quite functional and usable in many ways.

A possible usage scenario is in a classroom demonstration. Scientific application such as the one presented in the current article, could, for example be hosted on a mobile web server (see Nokia mobile web server [13] for instance) and can be instantly downloaded by students attending the class. The attendees need not necessarily be residing in the same geographical location. Such applications can

especially be used to demonstrate a concept (in this case quantum chemistry) in an interactive, first hand, easily accessible and understandable manner. The obvious advantage over a desktop solution is that a mobile device such as the one used in this study is easily portable, far cheaper than a standard desktop, and in-turn promote an anytime anywhere access. The author feels it important to note here that a demonstration need not necessarily have a strong visual component. For instance, the current application presented in this article, mobihf, at present only has basic visualization support. However, using mobihf, one can succinctly demonstrate various stages involved in an HF calculation and what would one generally expect out of such a calculation.

Another scenario where such applications could be useful is for marketing purposes. One can demonstrate a few critical capabilities of a much larger suite on a small, accessible and handy device. Best of all, such applications can easily be shared with prospective clients for them to play around and decide on purchasing the real product. Of course, it would be needless to say that one would need careful thought and creativity to build such a compelling application demo.

To be of some serious use, compute intensive applications built on mobile platforms need considerable thought in two areas: a) Power efficient methods b) Substantial improvements in standards of optimizing compiler and math libraries for mobile platforms. A few researchers have already done work in power profiling scientific methods and compute intensive applications such as 3D gaming [14, 15]. However, applying this research in building applications for mobile devices would also need considerable tool support. Tools such as the Nokia energy profiler could be of immense use in such scenarios. It is however clear that a lot of work in this area needs to be done. It could be well argued that this is generally due to lack of demanding applications. With the coming of more efficient mobile processors, however, it might catalyze improvement in quality of the tool support.

On the mobile application front, there has already been a few initiatives recently on using mobile devices for social science research [16] and as a tool for collecting data during a scientific field study [17]. These combined with quick computational tools on highly potable mobile devices could well be harbinger of interesting applications in this space.

4. CONCLUSION

In conclusion, this article takes a look at the feasibility of running a compute intensive scientific application on mobile devices. In doing so, a full quantum chemistry code that solves Hartree-Fock equation was written for the S60 platform. In the process, it brings out various issues that are of interest to a larger community of researchers as well as some of the practical usage scenarios of such a port. The first roadblock in porting a scientific application to a smart phone form-factor device is support for a good mathematical library, which to the best of author's knowledge is absent. Another interesting aspect to be aware of is that the best desktop implementation of an algorithm may not necessarily be most power efficient one, and hence there would be a need to investigate more on this issue when porting computing intensive applications. Finally, a subtle but important difference is the user interface to such applications on mobile devices would primarily be GUI rather than command line interface largely due to their form factors.

Apart from merely porting a compute intensive application to a mobile platform, there is high degree of possibly of incorporating interactiveness from a number inbuilt accessories on smart phones. These would include, but not limited to variety of connectivity options (Bluetooth, WiFi or 3G), camera, accelerometer or a GPS receiver. Looking forward, it would be rather interesting to observe more and more compute intensive applications being made available in mobile space, possibly bringing up new usage scenarios and insights into effective ways for making them power friendly and distributed over a large network.

ACKNOWLEDGMENTS

The author is thankful to Rick Muller, developer of PyQuante for including mobihf in its source and Dr. Alistair Renderll for useful comments.

REFERENCES

[1]. C. E. Kozyrakis and D. A. Patterson, "A new direction for computer architecture research," *IEEE Computer*, vol. 31, no. 11, pp. 24–32, 1998.

[2]. Wiki information on mobile operating systems. [Online]. Available:http://en.wikipedia.org/wiki/Mobile_operating_system

[3]. Catalog of ARM CPUs. [Online]. Available: http://www.arm.com/products/CPUs

[4]. Nokia E51 device details. [Online]. Available: http://www.forum.nokia.com/devices/E51

[5]. A. Szabo and N. S. Ostlund, Morden Quantum Chemistry. McGraw-Hill, New York, 1989.

[6]. The Symbian platform information. [Online]. Available: http://www.symbian.org/

[7]. PyS60: Python on S60 platform. [Online]. Available: http://wiki.opensource.nokia.com/projects/PyS60

[8]. mobihf: Quantum chemistry on mobile phones, based on PyQuante. [Online]. Available: http://tovganesh.googlepages.com/s60

[9].S60 SDK. [Online]. Available: http://developer.symbian.com/main/tools_and_sdks/sdks/

[10]. EMSL basis set exchange library. [Online]. Available: https://bse.pnl.gov/bse/portal

[11]. M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, *J. Comput. Chem.*, vol. 14, p. 1347, 1993.

[12]. The GAMESS package. [Online]. Available: http://www.msg.chem.iastate.edu/gamess/

[13]. Web server on S60 platform. [Online]. Available: http://research.nokia.com/research/projects/mobile-web-server/

[14]. X. Feng, R. Ge and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems" *Parallel and Distributed Processing Symposium*, vol. 1, p. 34, 2005.

[15]. B. Machocki, K. Lahiri and S. Cadambi, "Power analysis of mobile 3D graphics", DATE '06: Proceedings of the conference on Design, automation and test in Europe, pp 502, 2006.

[16]. Raento, A. Oulasvirta and N. Eagle, "Smartphones: Social Methods & Research", *Sociological Methods & Research*, vol. 37, No. 3, 426-454, 2009.

[17]. D. M. Kennedy, "Case studies of mobile applications in scientific field studies ", elearn 2009 [Online]. Available: http://elearn2009.com/index/schedule/

V. Ganesh is currently a Postdoctoral researcher at Australian National University. Prior to that he did his Ph.D. in development of methods for ab initio treatment of large molecules from University of Pune. When he is not working on this opensource projects, he is usually writing on his Blog, listening to Indian Classical or traveling.