INSTITUTE FOR DEFENSE ANALYSES

Parallel Compilation on Virtual Machines in a Development Cloud Environment

David A. Wheeler

September 2013

IDA Document D-4996
Log: H 13-001206
Copy

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882

About This Publication
This work was conducted by the Institute for Defense Analyses (IDA) under contract N66001-11-C-0001, subcontract D6384-S5, "Homeland Open Security Technology (HOST)," for Georgia Tech Research Institute. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

I N S T I T U T E   F O R   D E F E N S E   A N A L Y S E S

IDA Document D-4996

Parallel Compilation on Virtual Machines in a Development Cloud Environment

David A. Wheeler

Executive Summary

Government employees and contractors cannot, in some cases, install software development tools on their local systems. For example, they may be using a mobile device that cannot adequately support software development tools, or security policy restrictions may inhibit installing these tools. Yet, many tasks require the creation or modification of software, so this constraint can inhibit effective use and maintenance of both open source software (OSS) and government off-the-shelf software (GOTS). If these constraints cannot be addressed, the result could be higher costs, longer delays, or even a failure to perform necessary government tasks. A possible solution is to create "development clouds" that support software development without requiring any tools specific to software development to be installed in the local system.

An additional argument for development clouds is that they could use parallel processing to greatly reduce compilation time, speeding development for some software. However, it would be wise to test whether this hypothesis is actually true. The National Institute of Standards and Technology (NIST) definition of cloud computing does not specify the use of virtual machines (VM), but in practice many cloud environments are implemented using VMs. We wanted to confirm that at least some VM environments actually support parallel processing for performing compilations, and that using them can (at least in some circumstances) significantly speed compilations compared to a single central processing unit (CPU) in a VM. Confirmation is particularly important if a private cloud is to be acquired, since private clouds can be expensive.

This document describes an IDA innovation lab (iLab) experiment that verifies that VM environments exist that support parallel processing for performing compilations, and that using them can (at least in some circumstances) significantly speed compilations compared to a single-CPU case. Using the default settings, it took 166.85 minutes to compile the Linux kernel version 3.10.5 with 1 CPU, compared to 28.50 minutes with 16 CPUs, resulting in a reduction of 138.35 minutes (over 2 hours) on average for each full compilation. Thus, 16 CPUs compiled the software 5.85 times faster compared to one CPU. An additional optimization (involving the "-j" flag) slightly decreased the average 16-CPU compilation time even further. The fact that parallelism can speed compilations is not new, but this experiment confirms that this is also true in at least one currently available environment that implements VMs.

Of course, actual compilation times would vary considerably depending on a variety of factors. The amount of recompilation required for a particular change, and hardware differences, can vary compilation time substantially. Few tasks (including compilation) are perfectly parallel, limiting how much gain parallelism can provide. Still, this experiment gives evidence that, in at least some situations, assigning multiple CPUs to a VM can significantly speed compilation. It also demonstrates how to test this hypothesis for other situations.

A system that can cost-effectively provide many CPUs to execute a VM, such as a development cloud environment, can significantly speed up compilations. Therefore, development cloud environments may be a useful approach for supporting software creation and maintenance.

Contents

Figures

Tables

1. Introduction

Government employees and contractors cannot, in some cases, install software development tools on their local systems. For example, they may be using a mobile device that cannot adequately support software development tools, or security policy restrictions may inhibit installing these tools. Yet, many tasks require the creation or modification of software, so this constraint can inhibit effect use and maintenance of both open source software (OSS) and government off-the-shelf software (GOTS). The result can be higher costs, longer delays, or even a failure to perform necessary government tasks.

A possible solution is to create "development clouds" that support software development without requiring any tools specific to software development to be installed in the local system. A development cloud could be an additional function of a software "forge." A forge enables collaborative development of software source code, but forges do not always directly support all the tasks needed for development (since historically these tasks were done on a user's own system). Note that a development cloud is an application-specific use of cloud computing as defined by National Institute of Standards and Technology (NIST) Special Publication 800-145 [NIST].

An additional argument for "development clouds" is that they could speed development for software that is compiled. Many language systems require compilation during development. The compilation process can produce efficient code, but it is notorious for taking a long time, and developers using such language systems must repeatedly perform compilation. There are even cartoons pointing out that compilation can take a long time and that this time affects developers [Adams] [Cornet] [xkcd]. Many systems have been developed over the years to perform compilation in parallel, e.g., distributing the process across a network (using tools like distcc [Bonney]) or simply running many processes simultaneously when multiple processors are available. Since a large number of higher-performance processors can be aggregated on a development cloud, a cloud could in theory use many processors to compile a program. In particular, a cloud environment could provide far more Central Processing Units (CPU) than would typically be available on a desktop or laptop. If development clouds could significantly shorten compilation time, this might accelerate the use and acceptance of them by developers, since they would have a personal incentive to use development clouds. It would also simplify the use of recompilation to detect certain kinds of attacks [Wheeler].

Using development clouds as a solution seems reasonable, but before spending significant money or creating policies on cloud services, it would be wise to test this assumption. The NIST definition of cloud computing does not specify the use of virtual machines (VM), but in practice many cloud environments are implemented using VMs. A VM infrastructure adds new components that might fail to adequately support the use of parallel CPUs or might interfere with the ability to parallelize tasks. We wanted to confirm that at least some VM environments actually support parallel processing for performing compilations, and that using them can (at least in some circumstances) significantly speed compilations compared to a single-CPU case. Confirmation is particularly important if a private cloud is to be acquired, since private clouds can be expensive.

The experiment described in this paper does not commit the Government to any course of action (including the use, or non-use, of development clouds). Instead, it simply provides the Government with experimental data to help it understand the potential impact of a possible course of action.

2. Approach

We performed a simple experiment to determine whether the multiple CPUs available on cloud-based systems can significantly reduce compilation time.

We used the existing IDA innovation lab (iLab) equipment, which hosts VMs. This system is a Dell PowerEdge R720 with two Intel Xeon processors, each with eight cores. Each Intel Xeon processor was an E5-2650 2.00GHz, 20M Cache, 8.0GT/s QPI Turbo, with 96 GB of RAM. All VMs run on top of Citrix XenServer Enterprise Edition.

We set up a VM specifically for this experiment. This VM ran the widely used Fedora Linux distribution, version 19, including necessary compilation tools such as gcc.

We timed a compilation sample when one CPU was assigned to the VM, and compared that to timings when 16 CPUs were assigned to exactly the same VM (16 is the maximum number that can be assigned to a VM in the selected configuration). Since we used the same hardware, varying its use between 1 and 16 virtual CPUs, we did not have to worry about performance variation between different kinds of hardware. No other VMs were running on the relevant portion of the IDA iLab system while the tests were running.

We chose, as our compilation sample, the Fedora version of the Linux kernel version 3.10.5 (this is the same version that is used by Fedora 19 itself). We obtained its source code and Fedora build environment using the yum downloader tool. The Linux kernel was prepared (using “rpmbuild -bp kernel.spec”) and then compiled (built) from scratch. We performed this preparation and compilation five times in a row to get an average measure of time. All compilation caches (ccaches) were disabled, and only the compilation time was measured (as performed by “rpmbuild -bc --short-circuit --nocheck kernel.spec”). This is a large program; sloccount version 1.26 reports that this Linux kernel contains 11,256,252 physical source lines of code (SLOC), and that 97.27% is in the C programming language. Most of it (6,454,002 physical SLOC) is in its “drivers” subdirectory.

This is an intentionally extreme case:

1. We are recompiling the Linux kernel from scratch each time. In practice, software developer modifications often do not require complete recompilation of a system, and build dependency tools (such as make) and compilation caches (e.g., by ccache) can cause many unnecessarily recompilations to be skipped. However, since the amount of recompilation necessary greatly depends on the

specific modifications made, total recompilation from scratch is a reasonable representation of the worst case (from the point of view of a developer waiting for the process to complete). Also, many programs continue to use “recursive make”; a side effect of this common but poor practice is that recompilations often perform far more operations, and thus take more time, than might otherwise be expected [Miller].

2. While compilation is not perfectly parallelizable, in most cases compilation of large programs tends to be parallelizable. In many cases most files can be compiled independently, and only some tasks (e.g., linking of many files) are not parallelizable. It is already known that parallel recompilation on non-VM systems tends to significantly speed compilation.

3. The Linux kernel is a large program divided into a large number of separate files. Thus it provides many opportunities for parallel compilation.

We chose an intentionally extreme case because if a significant difference was not found between the use of 1 and 16 CPUs, given these circumstances, it is unlikely that a significant difference would be found in most other cases. We also timed cases while varying the number of CPUs between (and not including) 1 and 16; this enabled us to confirm the results, as well as help characterize how compilation time varies with the number of CPUs.

Our hypothesis was that assigning multiple CPUs to the VM should provide a significant decrease in the time to complete compilation compared to a single CPU, particularly given this intentionally extreme case. Of course, the whole point of an experiment is to test the hypothesis.

3. Findings

In this environment and configuration, multiple CPUs did significantly reduce compilation time (as expected). In the default configuration the average compilation time for 16 CPUs was 28.50 minutes, while for a single CPU it was 166.85 minutes. Thus, there was a savings of 138.35 minutes (over 2 hours) for each full compilation in the 16-CPU case compared to the single-CPU case.

Parallel processing speedup (one word) is defined as (time for 1 CPU/time for this many CPUs), and parallel processing efficiency is defined as (speedup/number of processors) [Eager]. Informally, speedup measures how many times faster many CPUs can solve some problem when compared to using a single CPU. In the 16 CPU case, with the default configuration, the speedup was 5.85 and the efficiency was 37%. This efficiency is nowhere near the ideal of 100%, but developers would still be delighted to reduce each compilation time by more than 2 hours.

As shown in the following figure, the speedup increases sublinearly as the number of CPUs increases.

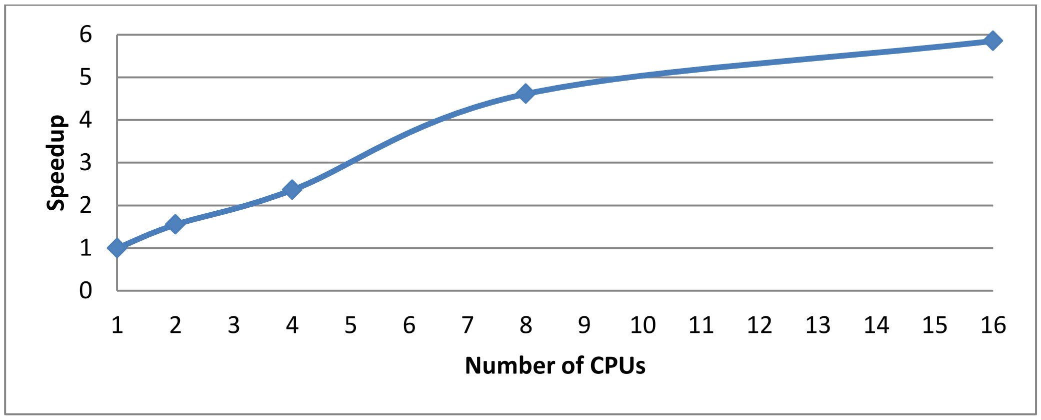

Figure 1. Average speedup varying number of CPUs, default configuration

The sublinear increase in speedup indicates that there is a decreasing efficiency in CPU use as the number of CPUs increases. This is a typical circumstance in parallel computing. The following figure shows this decreasing efficiency.

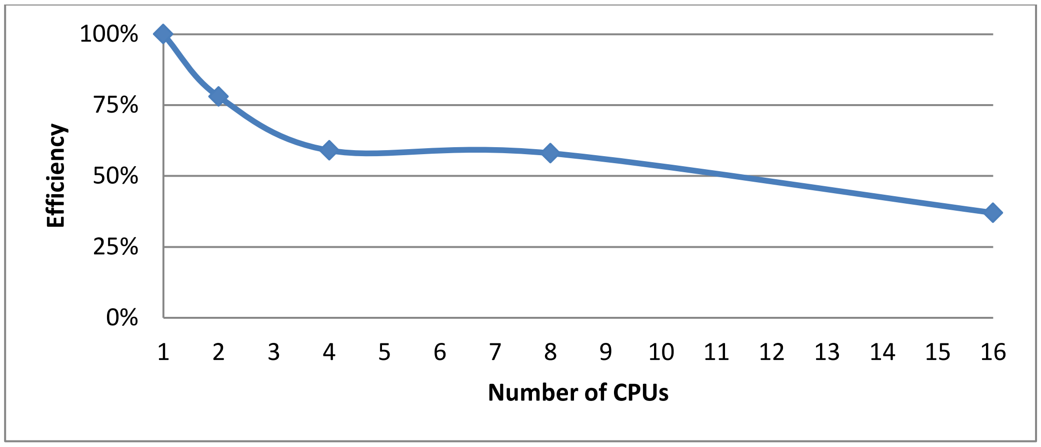

Figure 2. Average efficiency varying number of CPUs, default configuration

The "make" build program supports a "-j" option that can affect the number of simultaneous jobs [FSF]. We also ran the test cases while setting MAKEFLAGS to "-j *number*" and varying that number for both 1 CPU and 16 CPUs. In some cases this reduced the compilation time further, e.g., in the 16 CPU case, setting the "-j" option to 16 further reduced the average time to 23.86 minutes (compared to 28.50 when the option is not set), producing a speedup of 6.99 and efficiency of 44%. Of the values of -j tested, the best result was achieved with multiple CPUs when the number matched the number of processors (this is consistent with the experimental results of [Blaess]). This flag's effectiveness was (as expected) completely dwarfed by the effect of having (or not having) multiple CPUs.

The appendix provides more details on the experimental results.

4. Conclusions

Our findings show conclusively that, in at least some circumstances, virtual machines provided with many CPUs can perform compilations in significantly less time than a VM with one CPU. Using the default configuration it took 166.85 minutes with 1 CPU, compared to 28.50 with 16 CPUs. The fact that parallelism can speed compilations is not new, but we have confirmed that this is also achievable in VMs.

Of course, actual compilation times would vary considerably depending on a variety of factors. The amount of recompilation required for a particular change, and hardware differences, can vary compilation time substantially. Few tasks (including compilation) are perfectly parallel, limiting how much gain parallelism can provide. Configuration options can also affect efficiency (e.g., by modifying the “-j” option through MAKEFLAGS). Of course, the number of CPUs used also matters. Still, this experiment gives evidence that, in at least some situations, assigning multiple CPUs to a VM can significantly speed compilation. This experiment also demonstrates how to test this hypothesis for other situations.

There are many opportunities for future work. It would be interesting to investigate further how the number of CPUs affects compilation times. There are tools (e.g., distcc) that distribute compilation across a network, specifically to speed up compilation; we could test those running on VMs as well. We have every reason to believe that we could apply distributed compilation across a network with VMs, but our experiment did not test this. The iLab system has a second server with 32 additional CPUs; future experiments could determine whether adding those resources would speed compilation further, or conversely, whether using other VMs on the server (with different CPUs assigned to them) would cause a significant increase in the compilation time. It might be possible to modify the Linux kernel build system to be far more parallelizable; we have not investigated this. Instead of measuring recompilation time from scratch, we could measure average recompilation times after typical changes (e.g., by using version control system data as samples of typical changes). We could also recompile the program in a different way (e.g., not through rpmbuild), or compile different programs, to determine how these changes affect the results.

A system that can cost-effectively provide many CPUs to execute a VM, such as a development cloud environment, can significantly speed up compilations. Therefore, development cloud environments may be a useful approach for supporting software creation and maintenance.

Appendix A
Detailed Data

The following table shows the compilation times while varying the number of virtual CPUs, using a default configuration (MAKEFLAGS was left unset when compilation began). The table shows the average (mean) time (in minutes), the sample standard deviation, average speedup, average efficiency and actual times, when varying the number of CPUs along powers of 2 (speedup is time for 1 CPU/time for this many CPUs, and efficiency is speedup/number of processors [Eager]).

Table A-1. Compilation times varying number of CPUs, default configuration

CPUs	Average time (minutes)	Sample standard deviation	Speed-up	Effici-ency	Actual times
1	166.85	0.48	1	100%	166m5.906s, 167m24.466s, 166m55.956s, 167m1.046s, 166m47.885s
2	107.63	0.44	1.55	78%	107m2.299s, 107m19.588s, 107m43.468s, 107m59.898s, 108m2.966s
4	70.78	0.35	2.36	59%	70m25.171s, 70m25.326s, 70m50.348s, 71m1.050s, 71m11.496s
8	36.20	0.27	4.61	58%	35m46.956s, 36m30.632s, 36m13.059s, 36m11.372s, 36m18.051s
16	28.50	0.16	5.85	37%	28m41.823s, 28m37.153s, 28m26.343s, 28m18.840s, 28m25.350s

The following table shows the effect of varying the MAKEFLAGS value when the number of virtual CPUs varies between 1 and 16, with the average time, sample standard deviation (SStddev), and the actual compilation times.

Table A-2. Compilation times between 1 and 16 CPUs, varying MAKEFLAGS

	Time (minutes) for CPUs=1			Time (minutes) for CPUs=16		
MAKE FLAGS	**Aver-age**	**SStd dev**	**Actuals**	**Aver-age**	**SStd dev**	**Actuals**
(Unset)	166.85	0.48	166m5.906s, 167m24.466s, 166m55.956s, 167m1.046s, 166m47.885s	28.50	0.16	28m41.823s, 28m37.153s, 28m26.343s, 28m18.840s, 28m25.350s
-j 1	167.21	0.50	166m31.060s, 166m57.618s, 167m38.457s, 167m11.503s, 167m43.635s	28.97	0.27	29m16.752s, 28m58.992s, 28m50.076s, 29m8.809s, 28m35.256s
-j 2	174.75	1.45	173m50.743s, 173m51.798s, 174m29.776s, 174m13.211s, 177m18.070s	25.46	0.36	25m35.657s, 25m31.154s, 25m58.167s, 25m9.193s, 25m4.412s
-j 16	193.39	0.42	193m38.164s, 193m19.189s, 192m42.683s, 193m30.181s, 193m47.256s	23.86	0.36	24m10.411s, 23m46.658s, 24m17.934s, 23m29.803s, 23m34.281s
-j 32	207.50	0.53	208m19.729s, 207m14.792s, 207m21.893s, 206m54.724s, 207m40.279s	24.26	0.08	24m15.226s, 24m16.853s, 24m22.579s, 24m9.731s, 24m13.738s

Note that we time the compilation as performed by “rpmbuild”; this includes many tasks including key generation.

Also note that we are using samples of a larger population (of potentially-performed compilations), therefore, we use the sample standard deviation:

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x - \bar{x})^2}.$$

The sample standard deviation was calculated using the following Python code:

```
# Make code work on both Python 2.7 and the Python 3 series:
from __future__ import print_function, unicode_literals
from __future__ import division, absolute_import

from math import sqrt

def average_list(x):
  return sum(x) / len(x)

def sample_stddev(x): # Compute sample standard deviation of list
  average = average_list(x)
  sum_squares = sum( [(v-average)**2 for v in x] )
  return sqrt(sum_squares / (len(x)-1))
```

References

[Adams] Adams, Scott. 2013-06-22. *Dilbert*. http://dilbert.com/2013-06-22/

[Blaess] Blaess, Christophe. *Parallelizing Compilations*. 2012-01-14. http://www.blaess.fr/christophe/2012/01/14/parallelizing-compilations/

[Bonney] Bonney, Laurence. “Reduce compile time with distcc.” *IBM developerWorks*. 2004-06-22. http://www.ibm.com/developerworks/linux/library/l-distcc/index.html

[Cornet] Cornet, Manu. “One Last Time.” *Bonkers World* (sic). http://www.bonkersworld.net/one-last-time/

[Eager] Eager, Derek L, John Zahorjan, and Edward D. Lazowska. “Speedup Versus Efficiency in Parallel Systems.” *IEEE Transactions on Computers*, Vol. 38, No. 3, March 1989, pp. 408–423.

[FSF] Free Software Foundation (FSF). GNU Make Manual. July 28, 2010. http://www.gnu.org/software/make/manual/

[Miller] Miller, Peter. “Recursive Make Considered Harmful.” *Journal of Australian UNIX and Open Systems Users Group (AUUG)*, 19(1), pp. 14–25. http://miller.emu.id.au/pmiller/books/rmch/

[NIST] Mell, Peter, and Timothy Grance. *The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST)*. NIST Special Publication 800-145. September 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[Wheeler] Wheeler, David A. *Fully Countering Trusting Trust through Diverse Double-Compiling*. 2009. http://www.dwheeler.com/trusting-trust

[xkcd] Munroe, Randall. “Compiling.” *xkcd* (sic). http://xkcd.com/303/ Note: “xkcd” in all lower case is the preferred form, and is not an acronym; see http://xkcd.com/about/

Trademark owners are the owners of their respective trademarks. For example, Linux® is the registered trademark of Linus Torvalds in the United States and other countries. URLs are subject to change.

Acronyms and Abbreviations

ccache	Compilation Cache
CPU	Central Processing Unit
FSF	Free Software Foundation
GNU	GNU’s Not Unix
GOTS	Government off-the-Shelf
IDA	Institute for Defense Analyses
NIST	National Institute of Standards and Technology
OSS	Open Source Software
SLOC	Source Lines of Code
URL	Universal Resource Locator
VM	Virtual Machine

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE (DD-MM-YY) 13-09-2013	2. REPORT TYPE Final	3. DATES COVERED (From – To)
4. TITLE AND SUBTITLE Parallel Compilation on Virtual Machines in a Development Cloud Environment		5a. CONTRACT NUMBER N66001-11-C-0001, subcontract D6384-S5 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBERS
6. AUTHOR(S) David A. Wheeler		5d. PROJECT NUMBER 5e. TASK NUMBER GT-5-3329 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES Institute for Defense Analyses 4850 Mark Center Drive Alexandria, VA 22311-1882		8. PERFORMING ORGANIZATION REPORT NUMBER D-4996 H13-001206
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Joshua L. Davis Georgia Tech Research Institute 250 14th Street NW, Room 256 Atlanta, GA 30318		10. SPONSOR'S / MONITOR'S ACRONYM GTRI 11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

Project Leader: David A. Wheeler

14. ABSTRACT

Government employees and contractors cannot, in some cases, install software development tools on their local systems. For example, they may be using a mobile device that cannot adequately support software development tools, or security policy restrictions may inhibit installing these tools. A possible solution is to create "development clouds" that support software development without requiring any tools specific to software development to be installed in the local system. An additional argument for development clouds is that they could use parallel processing to greatly reduce compilation time. However, it would be wise to test whether this hypothesis is actually true for virtual machine (VM) environments, since they are often used to implement clouds.

This document describes an IDA innovation lab (iLab) experiment that verifies that VM environments exist that support parallel processing for performing compilations, and that using them can (at least in some circumstances) significantly speed compilations compared to a single-CPU case. Using the default settings, it took 166.85 minutes to compile the Linux kernel version 3.10.5 with 1 CPU, compared to 28.50 minutes with 16 CPUs, resulting in a reduction of 138.35 minutes (over 2 hours) on average for each full compilation. Thus, 16 CPUs compiled the software 5.85 times faster compared to one CPU. Additional optimizations (involving the "-j" flag) slightly decreased the average 16-CPU compilation time even further.

This shows that a system that can cost-effectively provide many CPUs to execute a VM, such as a development cloud environment, can significantly speed up compilations. Therefore, development cloud environments may be a useful approach for supporting software creation and maintenance.

15. SUBJECT TERMS

Parallel compilation, virtual machine, VM, development cloud, cloud, security, mobility, speed, speedup, software development, make, software creation, software maintenance, government off-the-shelf, GOTS, open source software, OSS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Joshua L. Davis
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	Unlimited	26	19b. TELEPHONE NUMBER (Include Area Code) 678-831-0182