Quantum Guarded-Command Language (qGCL) for Maximum Value

Aisya M. Suratno Putri[1], Jullia Titaley[2], Benny Pinontoan[3*]

[1,2,3]**Program Studi Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sam Ratulangi Manado**

*corresponding author email: bpinonto@yahoo.com

Abstract

On a classical computer or a binary computer, calculations are done simultaneously so as to produce the equations and algorithms. The result of this research shows that to determined maximum value specified in the algorithm using quantum Guarded-Command Language (qGCl) in quantum computer. Initially determine of maximum value was construct in Djikstra's Guarded-Command Language (GCL) which is then implemented on Zuliani's probability Guarded-Command Language (pGCL) furthermore applying to quantum Guarded-Command Language (qGCL) for last result. Of concern here is the speed in resolving a problem or calculate problem. Due to the Quantum Computer has a Quantum Bit (qubit) and a phenomenon commonly called superposition.

Keywords: GCL, pGCL, qGCL, quantum computer.

Quantum Guarded-Command Language (qGCL) untuk Nilai Maksimum

Abstrak

Pada komputer klasik atau komputer biner, perhitungan dilakukan secara simultan sehingga menghasilkan persamaan dan algoritma. Hasil penelitian ini menunjukkan bahwa nilai maksimum yang ditentukan dalam algoritma menggunakan quantum Guarded-Command Language (qGCl) dalam kuantum komputer. Awalnya menentukan nilai maksimum dibentuk dalam Guarded-Command Language (GCL) milik Djikstra yang kemudian diimplementasikan pada probability Guarded-Command Language (pGCL) milik Zuliani selanjutnya diterapkan pada quantum Guarded-Command Language (qGCL) untuk hasil akhir. Perhatian di sini adalah kecepatan dalam menyelesaikan masalah atau menghitung masalah. Karena Komputer Quantum memiliki Quantum Bit (qubit) dan fenomena yang biasa disebut superposisi.

Kata kunci: GCL, pGCL, qGCL, komputer quantum.

1. Introduction

In this era of science and technology, computer is well known to almost everyone around the world. Computer is known for it's performance, efficiency and effectiveness for computing something. But, despite of it's great usefulness, according to research about computers that have been carried out by scientist from the disciplines of mathematic, physic and computer science, this "currently-in-use" computers component are rapidly shrinking. In other words, it will running out of resources for its electronic circuits. In order to solve this problem, there is quantum computer that works based on the quantum physics phenomenon. The study and development had been carried out [1]. A guarded command is synonymous with a conditionally executed statement [2]. More precisely, a guarded command is the combination of a Boolean expression B and the statement S whose execution is controlled by B. in a sense, B "guarded" the execution of S. in Dijkstra's notation, a guarded command is represented as $\rightarrow S$.

In this research will use quantum Guarded-Command Language (qGCL) to build the simple computer program of quantum computers. It combines programming concepts in a compact way, before the program is written in some practical programming language. Its simplicity makes proving the correctness of programs easier.

2. Quantum Computer

Quantum computers utilizing the phenomenon of 'strange' is called a superposition. In quantum mechanics, a particle can be in two states at once. This is called a superposition state. In a quantum computer, in addition to 0 and 1 also known as a superposition of both. This means that the situation can be 0 and 1, instead of only 0 or 1 as in a normal digital computer. Quantum computer not using Bits but QUBITS (Quantum Bits). Because of its ability to be in various circumstances (multiple states), quantum computers have the potential to carry out a variety of calculations simultaneously so much faster than digital computers.

2.1. Quantum Bit (Qubit)

Qubit or quantum bit is a unit of quantum information, the quantum analogue of the classical bit, with additional dimensions associated to the quantum properties of the physical atom. The physical construction of a quantum computer itself is an arrangement of entangled atoms, and the qubit represents both the state memory and the state of entanglement in a system. A quantum computation is performed by initializing a system of qubits with a quantum algorithm. In a classical system, a bit would have to be in one state or the other, but quantum mechanics allows the qubit to be in a superposition of both states at the same time, a property which is fundamental to quantum computing.

2.2. Qubit State

A pure qubit state is a linear superposition of those two state. This means that the qubit can be represented as a linier combination of $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = a_0|b_0\rangle + a_1|b_1\rangle$$

Where:
$|b_0\rangle$ = qubit eigenstate of $0 = |0\rangle$
$|b_1\rangle$ = qubit eigenstate of $1 = |1\rangle$
$|\psi\rangle$ = qubit total state
a_0, a_1 = probability amplitudes and can in general both be complex numbers.

To measure this qubit in the standard basis, the probability of outcome $|0\rangle$ is $|a_0|^2$ and the probability of outcome $|1\rangle$ is $|a_1|^2$. Because the absolute square of the amplitudes equate to probabilities, it follows that α and β must be constrained by the equation$|a_0|^2 + |a_1|^2 = 1$ [3].

2.3. Quantum Programming

Quantum programming is a set of computer programming languages that allow the expression of quantum algorithms using high-level constructs. The point of quantum languages is not so much to provide a tool for programmers, but to provide tools for researchers to understand better how quantum computation works and how to formally reason about quantum algorithms.

2.3.1. Probabilistic Language (pGCL)

A guarded command language program is a sequence of assignment, skip and abort manipulated by the standard constructors of sequential composition, conditional selection, repetition and non deterministic choice. The BNF syntax of pGCL is as follows [4]:

$\langle program \rangle$ ::= {$\langle proc\ declaration \rangle$|; }$\langle qstatement \rangle${;
$\langle qstatement \rangle$}
$\langle statment \rangle$ ::= **Skip** |
Abort |
$x := e$|
$\langle proc\ call \rangle$ |
$\langle loop \rangle$ |
$\langle conditional \rangle$ |
$\langle nondeterministic\ choice \rangle$ |
$\langle probabilistic\ choice \rangle$ |
$\langle local\ block \rangle$
$\langle loop \rangle$ ::= **while** $\langle cond \rangle$ **do** $\langle statement \rangle$ **od**

$$\langle cond\rangle ::= \langle boolean\ expression\rangle$$
$$\langle conditional\rangle ::= \langle statement\rangle \lhd \langle cond\rangle \rhd \langle statement\rangle$$
$$\langle nondeterministic\ choice\rangle ::= \langle statement\rangle \Box \langle statement\rangle$$
$$\langle probabilistic\ choice\rangle ::= \langle statement\rangle\ {}_p\oplus\ \langle statement\rangle$$
$$\langle local\ block\rangle ::= \textbf{var}\bullet \langle statement\rangle\ \textbf{rav}$$

$$\langle proc\ call\rangle ::= \langle identifier\rangle(\langle actual\ parameter\ list\rangle)$$
$$\langle proc\ declaration\rangle ::= \textbf{proc}\langle identifier\rangle(\langle formal\ parameter\ list\rangle) \mathrel{\widehat{=}} \langle statement\rangle$$

2.3.2. Quantum Language qGCL (quantum Guarded-Command Language)

Quantum procedures can be of three kinds: *Initialisation* (or state preparation) followed by *Evolution* and finally by *Finalisation* (or observation) [5].

Initialisation is a procedure which simply assigns to its qureg state the uniform square convex combination of all standard states

$$\forall \chi : q(\mathbb{B}^n) \bullet \mathbf{In}(\chi) \mathrel{\widehat{=}} \left(\chi := \tfrac{1}{\sqrt{2^n}} \textstyle\sum_{x:\mathbb{B}^n} \delta_x\right)$$

Evolution models the evolution of quantum system and consists of iteration of unitary transformations on quantum state.

$$\forall \chi, \psi : q(\mathbb{B}^n) \bullet \langle \mathrm{U}(\chi), \mathrm{U}(\psi)\rangle = \langle \chi, \psi\rangle$$

In qGCL evolution is modeled via assignment: for example, $\chi := \mathrm{U}(\chi)$ models the evolution of qureg χ by means of unitary transform U.

Finalization is entirely defined using the probabilistic combinatory of pGCL :

$$\mathbf{Fin}[\mathcal{O}](i,\chi) \mathrel{\widehat{=}} \bigoplus \left[\left(i, \chi := j, \frac{P_{S_j}(\chi)}{\left\|P_{S_j}(\chi)\right\|}\right) @ \langle \chi, P_{S_j}(\chi)\rangle \mid 0 \le j < m\right]$$

2.3.3. Valid qGCL (quantum Guarded-Command Language)

The formal syntax for qGCL is as follow [4]:

$$\langle qprogram\rangle ::= \{\langle qproc\ declaration\rangle | \langle proc\ declaration\rangle;\}\ \langle qstatement\rangle\{;\langle qstatement\rangle\}$$
$$\langle qstatment\rangle ::= \chi := \langle unitary\ op\rangle(\chi) | \mathbf{Fin}(\langle identifier\rangle, [\langle identifier\rangle | \langle identifier\rangle, \langle identifier\rangle]) | \mathbf{In}(\langle identifier\rangle) | \langle identifier\rangle$$
$$\chi ::= \langle identifier\rangle$$
$$\langle qproc\ declaration\rangle ::= \textbf{qproc}\ \langle identifier\rangle(\langle formal\ parameter\ list\rangle) \mathrel{\widehat{=}} \langle qproc\ body\rangle$$
$$\langle qproc\ body\rangle ::= \textbf{var}\bullet\langle qproc\ statement\rangle\{;\langle qproc\ statement\rangle\}\textbf{rav}$$
$$\langle qproc\ statement\rangle ::= \textbf{skip}\ \ |x := e| \langle qloop\rangle | \langle qcondition\rangle$$
$$\langle qloop\rangle ::= \textbf{while}\ \langle cond\rangle\ \textbf{do}\ \langle qproc\ statement\rangle\ \textbf{od}$$
$$\langle qconditional\rangle ::= \langle qproc\ statement\rangle \lhd \langle cond\rangle \rhd \langle qproc\ statement\rangle$$

3. Guarded Command Language (GCL)

The guarded command is the most important element of the guarded command language. In a guarded command, just as the name says, the command is "guarded". The guard is a proposition, which must be true before the statement is executed. At the start of that statement's execution, one may assume the guard to be true. Also, if the guard is false, the statement will not be executed. The use of guarded commands makes it easier to prove the program meets the specification. The statement is often another guarded command [6].

Let $S, S_0, S_1, \ldots$ be statement, E be an expression, $B, B_0, \ldots$ be Boolean expression, x be any identifier and T be any type. Then, the syntax of statement in GCL is defined as follow:

$S ::= skip$		no-op
|	$x := E$	assignment
|	$S_1 : S_2$	sequencing
|	$if\ B_0 \rightarrow S_0 [] B_1 \rightarrow S_1 [] \ldots [] B_n \rightarrow S_n fi$	selection / condition
|	$do\ B \rightarrow S\ od$	iteration / repetitive
|	$[var\ x : T; S]$	block

4. Boolean –Valued Function

The name "boolean function" comes from the boolean logic invented by George Boole (1815-1864), an English mathematician and philosopher. As this logic is now the basic of modern digital computers, Boole is regarded in hindsight as a forefather of the field of computer science. Boolean values (or bits) are number 0 and 1. A boolean function $f(x) = f(x_1, \ldots, x_n)$ of n variables is a mapping $f : \{0,1\}^n \rightarrow \{0,1\}$. One says that f accepts a vector $\in \{0,1\}^n if\ f(a) = 1$, and rejects it if $f(a) = 0$. In other words, a Boolean-valued function is a function of the type $f : X \rightarrow \mathbb{B}$, where X is an arbitrary set and where $\mathbb{B}$ is a Boolean domain, where $\mathbb{B} = \{0,1\}$ whose element are interpreted as logical values, for example, 0=false and 1=true [7].

5. Hoare Logic

Hoare logic (also known as Floyd-Hoare Logic or Hoare rules) is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer program. The central feature of Hoare logic is the Hoare triple. A triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple is of the form

$$\{P\}S\{Q\}$$

Where :

- P and Q are assertions and S is a command.
- P is named the precondition and Q the postcondition

When the precondition is met, executing the command establishes the postcondition. Assertions are formulae in predicate logic. Hoare logic provides axioms and inference rules for all the constructs of a simple imperative programming language [8].

6. Research's Method

This research has been carried out by studying literature and applying it to make a simple computer program as an output. The variable used in this research will be listed associate to its theory. This research has been done with overall simple procedure that could be list by :

1. Literature collection and study. Collecting data (literatures about quantum computer especially it's properties and any algorithm that can be used in it) then studying it.
2. Construction Guarded Command Language. Make a simple case to construct guarded command. In this case, the author make a model about maximum value.
3. Applying simple Guarded Command Language on Quantum Computer using quantum Guarded-Command Language.

7. Result and Discussion

In this research, the author will make a simple GCL implemented into Quantum Computer using the semantic who it work in Quantum. The author using Djikstra's Guarded-Command Language and in quantum itself, the author using quantum Guarded-Command Language or better known as qGCL, which that qGCL is an extension of pGCL, which in turn extends Djikstra's Guarded-Command Language GCL with probabilism. But before that, the author will make the GCL in a classical computer.

7.1. Contsruct Simple Guarded Command Language

In the making of coding in Guarded Command is not too difficult, because essentially as same as using the pascal programming or the other. The first thing to do is to find some easy case or usual case

to make. After that, the author will make a model in GCL. This time, the author will make mathematical equations like addition and so forth. We know addition, subtraction, multiplication and division are the most basic things in mathematics. Because it can be easy to compare them in GCL programming. First of all, the author will try to form a GCL in a classical computer. The authors make three examples of different simple cases of GCL, to make it easier understood by the reader in constructing GCL, before the next stage of formation of cases used in this thesis is the maximum value.

Example 1:

In pseudocode :

$$if\ a < b\ then\ c \coloneqq True$$
$$else\ c \coloneqq False$$

In Guarded Command Language:

```
If a<b → c:=true
|  a≥b → c:=false
Fi
```

Example 1 above, using the condition rule, where the syntax using *if* and ends with *fi*.
In general from of this construct is:

$$if\ condition\ \rightarrow statement\ list$$
$$|\ condition\ \rightarrow statement\ list$$
$$|\ condition\ \rightarrow statement\ list$$
$$\ldots$$
$$fi$$

Each conditions of the above is known as the Guard. Guard and the following statement, together, called Guarded Command. when control reach if the statement in the language with guarded command, a nondeterministic choice made between guard that evaluates true, and statements the following list of selected Guarded be executed. The final condition may optionally be else. If none of the conditions evaluates to true, the statement list following the else, if any, is executed. If there is no else, the *if* statement as a whole has no effect.

Example 2:

After that the author will try to make using Skip command.
In pseudocode :

$$if\ error = True\ then\ x \coloneqq 0$$

In Guarded Command Language :

```
If error=True → x:=0
 | error=False Skip
Fi
```

In GCL there are two command called Skip and Abort. Skip and Abort command are very different. Abort is the undefined instruction: do anything. The abort statement does not even need to terminate. It is used to describe the program when formulating a proof, in which case the proof usually fails. Skip is the empty instruction: do nothing. It is used in the program itself, when the syntax requires a statement, but the programmer does not want the machine to change states. At this time, the authors use the Skip command just for an example. When the Skip command is done, the program will stop doing any command.

After the author tried in example 1 and 2 using the condition rule $if - fi$ and skip rule, the author tried again with another example. For a basic command such a sum in arithmetic and etc, it remains the same in the GCL. If we write any in pseudocode command such a addition (+), subtraction (-), multiplication (*) and division (/) then the GCL sheetscode remains the same.

Example 3:

In Pseudocode :

$z = x + y$ | $z = x * y$
$z = x - y$ | $z = x/y$

In Guarded Command Languge :

```
[[
Var
        Z := x+y
        Z := x-y
        Z := x*y
        Z := x/y
]]
```

Basically, the writing on the command guarded by pascal programming, so the coding was formed is not much different. According to GCL design in the above, there are no changes to be when pseudocode is made. But it was only a base before construct it into Guarded Command Language.

After the author construct a coding in GCL, the author implement or apply it into a quantum computer. In applying it, the author uses qGCL as mentioned earlier in this chapter (IV). But the author doesn't use the above example. The author want to using example to find or determine the maximum value. To see it in the qGCL form, the author using the model of maximum value for that GCL.

7.2. Determine The Maximum Value

In this section, the author will make coding for GCL (Guarded Command Language) about looking for maximum value. Model programs created using the *x* and *y* where *x* and *y* are integers which entered into coding assigns where *m* as the maximum value of *x* and *y* using condition rule.
In Pseudecode :

$$if\ x > y\ then\ m \coloneqq x\ else\ m \coloneqq y$$

In Guarded Command Language :

```
[[
Var m,x,y:int;
       If x>y → m:=x
       |  x<y → m:=y
       Fi
]]
```

7.3. Using Derivation of Program

With designs on Guarded Command on top of the maximum value, still can not be applied to quantum computers, because it was the author will do a formal derivation of the program.
The formal requirement of writer program performing is $\coloneqq \max(x, y)$. So with follows:

- **Precondition :**
  Precondition is a predicate describing the condition the function relies on for correct operation. So, the precondition of the perform is
  $$x \geq y \wedge y \geq x$$
- **Postcondition :**
  Postcondition is a predicate describing the condition the function establishes after correctly running. To see the perform construct above is that for fixed *x* and *y* it establishes the relation
  $$R : (m = x\ or\ m = y) \wedge m \geq x \wedge m \geq y.$$
  For first, let $m \coloneqq x$ for standard the way of establishing the truth of $m = x$ for fixed x. Take weakest precondition as it's guard. Deriving
  $$wp(\text{"m:=x"}, R) = (x = x\ or\ x = y)\ and\ x \geq x\ and\ x \geq y$$
  $$= x \geq y$$
  For $\coloneqq y$, to make fixed for y with the assignment "$m \coloneqq y$" with the guard
  $$wp(\text{"m:=y"}, R) = y \geq x\ ;$$
  So, back to above perform, the author get,

```
       If x>y → m:=x
       |  y>x → m:=y
       Fi
```

- **Invariant**

  As an example of the derivation of a repetitive construct, author derive a program for the greatest common divisor (gcd) of two positive numbers for fixed positive x and y have to establish the final relation.

  $$x = \gcd(X, Y)$$

  For the invariant relation

  $$P\text{: } \gcd(X, Y) = \gcd(x, y)\ and\ x > 0\ and\ y > 0$$

  A relation that has the advantage of being easily established by $x := X; y := Y$. Take weakest precondition as it's guard, because about the gcd-fuction at which $\gcd(x, y) = \gcd(x - y, y)\ if\ x > y$, a possible guarded list would be $x := x - y$. Deriving

  $$wp(\text{x:=x-y}), P = (\gcd(X, Y) = \gcd(x - y, y)\ and\ x - y > 0\ and\ y > 0)$$

  So the conjuction implied by P, and the guarde $x > y$ as far as the invariance of P is concerned. Then deriving

  $$wp(\text{y:=y-x}), P = (\gcd(X, Y) = \gcd(y - x, x)\ and\ y - x > 0\ and\ x > 0)$$

  And the guard would be $y > x$. The next effort is the construct would be this performs :

```
X:=X ; y:=Y
      do x>y → x:=x-y
      |  y>x → y:=y-x
      od
```

  And with the same version using Euclid's Algorithm in great common divisor, with would have written down with follows :

```
 X:=X ; y:=Y ;
While x ≠ y do if x > y then x := x - y
                             else y := y - x
                  fi
              od
```

7.4. Applying to qGCL (quantum Guarded Command Language)

In this section, the author was applying a program that has been made above that determine the maximum value added to quantum computers that where here the author uses qGCL. As in what is already known according to section 2.2.4 of the formal syntax for qGCL. In this section will using $\langle qloop \rangle$ and $\langle qconditional \rangle$ in a models.

Before construct into qGCL, the author will use the other Quantum Programming for probabilistic that is using pGCL. By using $\langle proc\ declaration \rangle$ and value parameter is $x, y: \mathbb{B}^k$ where x and y identifier as Boolean-valued functions on $0..k$ or called as bit register. Parameter result is χ where χ is a square-convex complex superposition of standart states.

Proc exp (value x,y:$\mathbb{B}^k$; result χ:$\mathbb{B}^n$) $\widehat{=}$ $statement$

With a statement that is formed according to the model in section 4.1.2, with syntax $\langle local\ block \rangle$ is **var•** $\langle statement \rangle$ **rav .** The statement in here is the body portion of the contents of the model with be created using loop rule that is **while** $\langle cond \rangle$ **do** $\langle statement \rangle$ **od,** then the model is formed to be like this below.

```
Proc exp (value x,y:𝔹^k ; result χ:𝔹^n) ≙
Var x,y •
χ := 0;
while x ≠ y do
              if x > y →  χ:= x - y
              |   y > x →  χ:= y - x
              fi
              od
rav
```

Because that terms of pGCL itself is expresses the qGCL, that enables to combine code and specification since the result has a semantic denotation to which refinement applies. pGCL denotes the guarded-command language extended to contain probabilism. So the next thing to do is to design

qGCL where the author was simply replace **proc** by **qproc** as in $\langle qproc\ declaration \rangle$ and let the compiler doing all the necessary type changes like in section 2.2.4. Procedure **exp** now become **qexp**. The author use procedure **exp** as a quantum procedure because it is possible to compute exponentiation on quregs in a superposition of standard states. The parameters used are value result for $x, y : \delta(\mathbb{B}^k)$ where x and y identifier as qureg (qubit registers).

qproc qexp (value result x,y:$\delta(\mathbb{B}^k)$; value result $\chi : \delta(\mathbb{B}^n)$) $\widehat{=}$ $statement$

in this case the author use a substraction operator $Sub(\chi, \psi)$ with initialitation $\chi \coloneqq 0$ is readily modelled via the Dirac δ map. For the overall design will be shown as below :

```
qproc qexp (value result x,y:δ(𝔹^k) ; value result χ:δ(𝔹^n)) ≙
Var x,y •
χ ≔ δ0;
while x ≠ y do
                if x > y →   χ ≔ Sub(x,y)
                |    y > x →   χ ≔ Sub(y,x)
                fi
                od
rav
```

In qGCL models above are very similar to those the author form in pGCL. In pGCL itself the Guarded-Command Language extended with probabilism. So that listed most common form GCL itself. And in qGCL, GCL combined with quantum computation logic. The program is modeled can be quickly calculated due qGCL utilize Dirac δ where on quantum programming Dirac δ is acting as a superposition.

In this study proves that in the quantum programming is needed GCL or so-called Guarded-Command Language. As in the form of pGCL with probabilism and qGCL are essentially using GCL. In other words GCL is in a classical computer language program that is used in solving problems in quantum computer before it is implemented.

8. Conclusion

Based on the result that has been described in the last section, obtained conclusions that GCL or called Guarded Command Language is easy to construct program because the essentially the same as using the pascal programming or the others. The term of this guarded command is simple. The guarded will be executed when its true. As has been done by the author in determining the maximum value.

In this research considered Djikstra's plain Guarded Command Language thus omitting probabilism, so the author expand the normal form approach from pGCL and in turn to qGCL.

If the GCL was proven is true, then implemented into the Quantum Computer will use pGCL or qGCL will be true. Of concern here is the speed in resolving a problem or calculate problem. Due to the Quantum Computer has a Quantum Bit (qubit) and a phenomenon commonly called superposition. This proves that the Quantum Computer itself is not only made to carry out an issue or program quickly but to solve the problem where the problem may not be solved in Classical computer (digital computer).

9. References

[1] Marinescu, D.C and G.M. Marinescu. 2010. *Classical and Quantum Information*. Academic Press.

[2] Dijkstra, E. W. 1975. Guarded Commands, Nondeterminacy and The Formal Derivation of Programs. *CACM*. 18:453-457.

[3] Williams, Collin P. and Clearwater, Scott H. 1998. *Explorations in Quantum Computing*. *Springer-Verlag*. New York.

[4] Zuliani, P. 2005. Compiling Quantum Programs. *Journal Acta Informatica*. 00: 1-39.

[5] Zuliani, P. 2004. Non-Deterministic Quantum Programming. *Proceeding QPL 2004. Facoltà di Scienze e Tecnologie Informatiche Libera Università di Bolzano Italy*. pp 179–195.

[6] Sanders, J. W. and P. Zuliani. 2000. Quantum programming. Mathematics of Program Construction, *Springer-Verlag LNCS*. 1837 : 84.

[7] Jukna, S. 2012. Boolean Function Complexity, Algorithms, and Combinatorics. *Springer-Verlag*. Berlin Heidelberg. 27 : 3-53.

[8] Ying, Mingsheng. 2013. Hoare Logic for Quantum Programs. *Samsonfest.* University of Technology. Sydney.