Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

03. Αρχιτεκτονική και απόδοση υπολογιστών

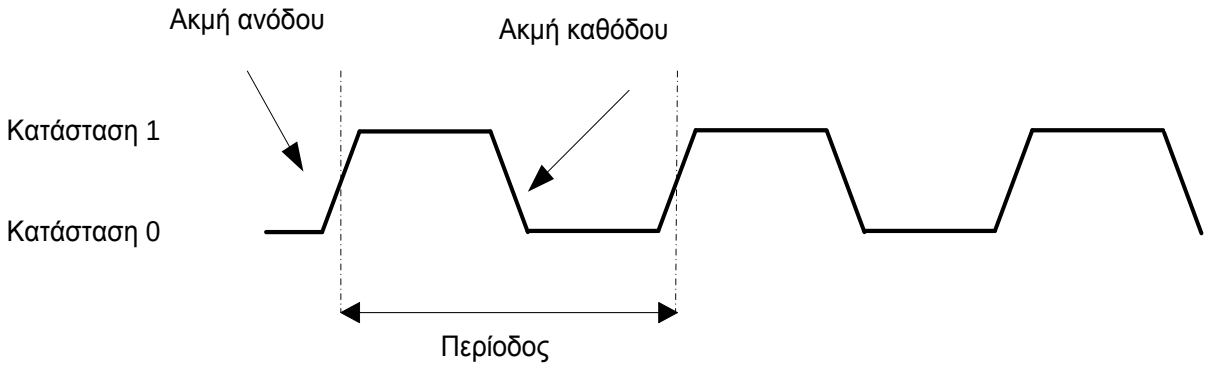
Αξιολόγηση μιας Αρχιτεκτονικής

- Αξιολόγηση Αρχιτεκτονικής =Αξιολόγηση της επίδοσης των υπολογιστών που την υλοποιούν.

- Συνήθεις μετρικές (metrics):
  - Χρόνος CPU.
  - Κύκλοι ανά εντολή (ή Εντολές ανά κύκλο).
  - Εκατομμύρια εντολές ανά δευτερόλεπτο (MIPS).
  - Εκατομμύρια πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο (MFLOPS).
  - Μετροπρογράμματα (Benchmarks).

Η απόδοση της CPU - Ορισμοί

- Κύκλος ρολογιού (Clock Cycle): Το διάστημα μεταξύ δύο γεγονότων ρολογιού (π.χ. ακμές ανόδου).
- Περίοδος κύκλου (Cycle Period): Η διάρκεια του κύκλου σε ns.
- Συχνότητα ρολογιού (Clock rate): Η συχνότητα των κύκλων στη μονάδα του χρόνου.
- [Συχνότητα ρολογιού] = 1 / [Περίοδος κύκλου] (GHz).



Παράδειγμα

- Περίοδος = 0.4ns. Σε πόσα GHz αντιστοιχούν;
- Συχν = συχνότητα ρολογιού = Clock Rate.
- Περ = Περίοδος κύκλου = Cycle Period.

$$\text{Συχν} = \frac{1}{\text{Περ}} = \frac{1}{0.4ns} = \frac{1}{0.4\times10^{-9}\,\text{sec}} = \frac{1}{4\times10^{-10}\,\text{sec}}$$

$$= \frac{10^{10}}{4}\frac{1}{\text{sec}} = 2.5\cdot10^{9}\,Hz = 2.5GHz$$

Χρόνος CPU

- **[Χρόνος CPU] = [Χρόνος CPU χρήστη] + [Χρόνος συστήματος]**
- Χρόνος συστήματος: Χρόνος που δαπανάται από το σύστημα για την διαχείριση του προγράμματος (δημιουργία της διεργασίας, απόδοση χώρου στη μνήμη, δρομολόγηση της διεργασίας για εκτέλεση, χρόνος αναμονής λόγω multitasking, κλπ). Αφορά στο λειτουργικό σύστημα.
- Χρόνος CPU χρήστη: Χρόνος που δαπανάται από την CPU για την πραγματική εκτέλεση του προγράμματος. Αφορά στο πρόγραμμα του χρήστη και την αρχιτεκτονική του Η/Υ.

Απόδοση (Performance) υπολογιστών

- Μια πιο ρεαλιστική προσέγγιση είναι να ορίσουμε την απόδοση (performance) του υπολογιστή X σαν το αντίστροφο του χρόνου εκτέλεσης.
- $[Απόδοση]_X = 1 / [Χρόνος\ εκτέλεσης]_X$
- Αυτό πρακτικά σημαίνει ότι η απόδοση του υπολογιστή *X* είναι *n* φορές μεγαλύτερη σε σχέση με τον *Y*, όταν:

$$n = \frac{[\text{Απόδοση}]_X}{[\text{Απόδοση}]_Y} = \frac{[\text{Χρόνος Εκτέλεσης}]_Y}{[\text{Χρόνος Εκτέλεσης}]_X}$$

Χρόνος CPU χρήστη (CPU time, 1)

- **[Χρόνος CPU χρήστη] = [Πλήθος εντολών] × [Μέσος χρόνος εκτέλεσης μιας εντολής]**
- Αφορά στο πρόγραμμα του χρήστη και την αρχιτεκτονική του Η/Υ.
- Πλήθος εντολών (Instruction Count – IC): το συνολικό πλήθος των εντολών του προγράμματος.
- [Μέσος χρόνος εκτέλεσης μιας εντολής] = [Πλήθος κύκλων ανά εντολή] x [Περίοδος κύκλου]
- Μέσο πλήθος κύκλων ανά εντολή (Cycles per Instruction – CPI): ο μέσος αριθμός των κύκλων που χρειάζεται μια εντολή γλώσσας μηχανής για να εκτελεστεί.
- Ανεξάρτητο από το πρόγραμμα και από τη συχνότητα του ρολογιού. Αφορά καθαρά την αρχιτεκτονική.

Χρόνος CPU χρήστη (CPU time, 2)

[Χρόνος CPU χρήστη] =
= [πλήθος εντολών] × [πλήθος κύκλων ανά εντολή] × [περίοδος κύκλου]

$$= \frac{[\text{πλήθος εντολών}] \times [\text{πλήθος κύκλων ανά εντολή}]}{[\text{συχνότητα ρολογιού}]}$$

Χρόνος CPU χρήστη (CPU time, 3)

CPU_{time} = Πλήθος εντολών (IC) ⟵ Instruction set Architecture + Τεχνολογία Compiler

× Πλήθος κύκλων ανά εντολή (CPI) ⟵ Instruction set Architecture + Οργάνωση

× Περίοδος Κύκλου (CP) ⟵ Τεχνολογία Hardware + Οργάνωση

CPU time = IC x CPI x CP

Ο γενικευμένος βασικός τύπος

$$CPU\ Time = \left(\sum_{i=1}^{N} IC_i \times CPI_i \right) \times Clock\ \ cycle\ time$$

$$CPI\ (average) = \frac{\sum_{i=1}^{N} IC_i \times CPI_i}{Instruction\ Count} = \sum_{i=1}^{N} \frac{IC_i}{Instruction\ Count} \times CPI_i$$

- Instruction Count = $IC = IC_1 + IC_2 + ... IC_N$
- Υποθέτουμε ότι ένα πρόγραμμα αποτελείται από N διαφορετικό πλήθος εντολών ή ομάδων εντολών γλώσσας μηχανής όπου IC_i παριστάνει το πλήθος των κύκλων της εντολής i στην οποία αντιστοιχεί το CPI_i.

Παράδειγμα 1

- Δύο μηχανές *A* και *B* έχουν την ίδια αρχιτεκτονική του instruction set αλλά διαφορετική υλοποίηση.
- Η μηχανή *A* έχει περίοδο κύκλου = 1ns και τρέχει ένα πρόγραμμα με CPI = 2.
- Η μηχανή *B* έχει περίοδο κύκλου = 2ns και τρέχει το ίδιο πρόγραμμα με CPI = 1.2.
- Ποια μηχανή έχει μεγαλύτερη απόδοση και πόσες φορές;

Λύση

- Ο αριθμός (έστω N) των εντολών του προγράμματος θα είναι ο ίδιος και στις δύο μηχανές λόγω της ίδιας αρχιτεκτονικής.

$$CPU\ Clock\ cycles_A = InstructionCount \times CPI_A = N \times 2.0$$
$$CPU\ Clock\ cycles_B = InstructionCount \times CPI_B = N \times 1.2$$

$$CPU\ time_A = CPU\ Clock\ cycles_A \times Clock\ Cycle\ time = N \times 2.0 \times 1ns = 2.0 \times N \quad ns$$
$$CPU\ time_B = CPU\ Clock\ cycles_B \times Clock\ Cycle\ time = N \times 1.2 \times 2ns = 2.4 \times N \quad ns$$

$$\frac{Performance_A}{Performance_B} = \frac{CPU\ time_B}{CPU\ time_A} = \frac{2.4 \times N \quad ns}{2.0 \times N \quad ns} = 1.2$$

- Απάντηση: Η απόδοση του Α είναι 1.2 φόρες μεγαλύτερη της απόδοσης του Β.

Παράδειγμα 2

- Μια μηχανή έχει instruction set το οποίο διαθέτει τρεις κατηγορίες εντολών (A, B και C) γλώσσας μηχανής. Κάθε κατηγορία έχει και διαφορετικό CPI (Cycles Per Instruction). Η κατηγορία A έχει CPI=1, η B έχει CPI=2 και η C έχει CPI=3. δηλαδή οι εντολές τις κατηγορίας A εκτελούνται σε ένα κύκλο της κατηγορίας B σε δύο κύκλους και της κατηγορίας C σε τρεις κύκλους.
- Ο σχεδιαστής ενός compiler μιας ανώτερης γλώσσας προγραμματισμού επιθυμεί να βρει την καλύτερη μέθοδο μετάφρασης την οποία και θα ενσωματώσει στον compiler. Πρέπει να διαλέξει μεταξύ δύο μεθόδων: την X και την Y. Καλύτερη θεωρείται η μέθοδος μετάφρασης η οποία παράγει για τον ίδιο πηγαίο κώδικα και για την ίδια CPU τον γρηγορότερο εκτελέσιμό κώδικά.
- Παίρνει ένα τυχαίο πρόγραμμα και το μεταφράζει με τη μέθοδο μετάφρασης X και μετά με τη μέθοδο μετάφρασης Y. Κάθε μέθοδος παράγει κώδικα ο οποίος έχει διαφορετικό instruction mix το οποίο φαίνεται στον παρακάτω πίνακα.

Παράδειγμα 2

	Πλήθος Εντολών Γλώσσας μηχανής (IC)		
	Κατηγορία A	**Κατηγορία B**	**Κατηγορία C**
Μέθοδος X	20	10	20
Μέθοδος Y	40	10	10

- Ποια μέθοδος θα προτιμηθεί ?

Λύση: Έλεγχος της μεθόδου Χ

$$CPU\ clock\ cycles = \sum_{i=1}^{N} IC_i \times CPI_i$$

- Επειδή IC_A=20, IC_B =10, IC_C= 20 και CPI_A=1, CPI_B=2, CPI_C =3 και
- **CPU Clock cycles** = 1*20 + 2*10 + 3*20 = 100 κύκλοι.
- **Instruction count** = 20 + 10 + 20 = 50 εντολές
- **CPI (average)** = 100/50 =2.0 κύκλοι/εντολή.
- ή **CPI (average)** = 20/50 *1 + 10/50*2 + 20/50*3 = 0.4+0.4+1.2= 2.0 κύκλοι/εντολή.

Λύση: Έλεγχος της μεθόδου Υ

$$CPU\ clock\ cycles = \sum_{i=1}^{N} IC_i \times CPI_i$$

- Επειδή IC_A=40, IC_B =10, IC_C= 10 και CPI_A=1, CPI_B=2, CPI_C =3 και
- **CPU Clock cycles** = 1*40 + 2*10 + 3*10 = 90 κύκλοι.
- **Instruction count** = 40 + 10 + 10 = 60 εντολές.
- **CPI (average)** = 90/60 = 1.5 κύκλοι/εντολή.
- ή **CPI (average)** = 40/60 *1 + 10/60*2 + 10/60*3 = 0.67+0.33+0.5 = 1.5 κύκλοι/εντολή.

Συμπέρασμα

- Ο σχεδιαστής θα προτιμήσει την μέθοδο που δημιουργεί εκτελέσιμο κώδικα ο οποίος τελειώνει γρηγορότερα στην ίδια CPU. Δηλαδή αυτή που μας δίνει μικρότερο CPU χρόνο ή μεγαλύτερη απόδοση.
  - CPU time (X) / CPU time (Y) =
    CPU cycles(X)*Clock time / CPU cycles(Y) * Clock time.
- CPU cycles(X) / CPU cycles(Y) = 100/90 = 1.1 .
- Άρα CPU time(X) = CPU time(Y) * 1.1 ή Απόδοση (Y) = Απόδοση (X) * 1.1.
- Άρα μέθοδος Y είναι η καλύτερη γιατί παρουσιάζει μεγαλύτερη απόδοση CPU.
  - Παρατήρηση : Το clock time είναι ίδιο γιατί πρόκειται για την ίδια CPU.

MIPS και MFLOPS

$$MIPS = 10^{-6} \frac{Πλήθος\ εντολών}{[Χρόνος\ εκτέλεσης]}$$

- MIPS = (Mega) Instructions Per Second
  - Μειονέκτημα: δεν διακρίνει τις εντολές ανάλογα με το χρόνο εκτέλεσής τους.

$$MFLOPS = 10^{-6} \frac{Πλήθος\ πράξεων\ κινητής\ υποδιαστολής}{[Χρόνος\ εκτέλεσης]}$$

- MFLOPS = (Mega) FLoating point Operations Per Second.
  - Μειονεκτήματα: όμοια με το MIPS. Επίσης είναι χρήσιμο κυρίως για επιστημονικές εφαρμογές που χρησιμοποιούν πολλές πράξεις κινητής υποδιαστολής.

Μετροπρογράμματα (Benchmarks)

- Πραγματικά προγράμματα, π.χ. μεταφραστές, επεξεργαστές κειμένου, εργαλεία CAD, κλπ.
- Πυρήνες προγραμμάτων, δηλαδή μικρά αποσπάσματα κώδικα που χρησιμοποιούνται συχνά από πολλά προγράμματα. Οι πυρήνες Livermore Loops και Linpack είναι τα πιο γνωστά παραδείγματα.
- Μικρά προγράμματα που όμως χρησιμοποιούντα συχνά, π.χ. Quicksort.
- Συνθετικά benchmarks, π.χ. τα Whetstone και Dhrystone, τα οποία έχουν την ίδια φιλοσοφία με τους πυρήνες και προσπαθούν να προσομοιώσουν το μίγμα των εντολών που παρουσιάζονται σε μεγάλες εφαρμογές.

SPEC Benchmarks

- Το Standard Performance Evaluation Corporation (SPEC) είναι ένας μη κερδοσκοπικός οργανισμός ο οποίος δημιουργεί, οργανώνει και συντηρεί πρότυπα benchmarks τα οποία εφαρμόζονται στους υπολογιστές προκείμενου να μετρήσουν την απόδοση τους.
- Ο SPEC εκτός από την ανάπτυξη των λεγομένων benchmark suites σχολιάζει και δημοσιοποιεί αποτελέσματα τα όποια στέλνονται σε αυτόν από τους κατασκευαστές
- http://www.spec.org/.

Κατηγορίες SPEC

- Αξιολόγηση CPU:
  - SPEC CPU2006
  - SPEC CPUv6
- High Performance Computing (MPI, OpenMP):
  - SPEC MPI2007
  - SPEC OMP2001
  - SPECjbb2005
  - SPECjEnterprise2010
  - SPECjms2007
  - SPECjvm2008
- Web Servers
  - SPECweb2009
  - SPECweb2005

Κατηγορίες SPEC

- Άλλες κατηγορίες:
  - Graphics/Applications
  - Java Client/Server
  - Mail Servers
  - Network File System
  - Power
- Network File System
  - SFS97_R1 (3.0)
  - SFS97 (2.0)
  - SFS93 (LADDIS)

SPEC CPU2006: Integer benchmarks

Όνομα	Γλώσσα	Περιγραφή
400.perlbench	C	PERL Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: go
456.hmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics: Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

SPEC CPU2006: Floating Point benchmarks

Όνομα	Γλώσσα	Περιγραφή
410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics: Quantum Chromodynamics
434.zeusmp	Fortran	Physics / CFD
435.gromacs	C/Fortran	Biochemistry/Molecular Dynamics
436.cactusADM	C/Fortran	Physics / General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology / Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C/Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrf	C/Fortran	Weather Prediction
482.sphinx3	C	Speech recognition

Dhrystone και Whetstone Benchmarks

- Dhrystone: Προτάθηκε το 1984 από τον R.P. Wecker. Είναι ένα benchmark program γραμμένο σε C, Pascal ή Java το οποίο αξιολογεί ένα σύστημα με βάση τη δυνατότητα του στους ακέραιους υπολογισμούς.
- Το πρόγραμμα εκτελεί μόνο ακέραιες πράξεις και δεν έχει καθόλου λειτουργίες εισόδου/εξόδου (I/O) ούτε κλήσεις προς το λειτουργικό σύστημα. Dhrystones per second είναι η αντίστοιχη μονάδα μέτρησης και εκφράζει το πλήθος των επαναλήψεων του προγράμματος σε ένα δευτερόλεπτο.
- Το αντίστοιχο πρόγραμμα για τις πράξεις κινητής υποδιαστολής ονομάζεται Whetstone.

Ο νόμος του Amdahl (1)

- Ένας μύθος:
- «Αν βάλω Ν επεξεργαστές να εκτελούν παράλληλα ένα πρόγραμμα τότε αυτό θα εκτελεστεί Ν φορές πιο γρήγορα».
- Δεν ισχύει διότι:
  - Έχουμε κόστος επικοινωνίας.
  - Ο αλγόριθμος σπάνια τεμαχίζεται σε Ν τελείως ίσα κομμάτια.
  - Σε κάθε αλγόριθμο υπάρχει ένα σειριακό κομμάτι που δεν παραλληλίζεται (Νόμος του Amdahl).

Ο νόμος του Amdahl (2)

- Ακόμα και αν αγνοήσουμε το χρόνο επικοινωνίας και αν υποθέσουμε ότι τεμαχίσαμε τον κώδικα σε απολύτως ίσα κομμάτια η επιτάχυνση που θα επιτύχουμε δεν είναι μεγαλύτερη από $1/s$ όπου s είναι το ποσοστό του αλγορίθμου που δεν παραλληλίζεται, δηλαδή το ποσοστό του σειριακού τμήματος:

$$Επιτάχυνση < 1/s$$

- Έτσι αν ο αλγόριθμος έχει $s = 0.1$ (δηλαδή $s = 10\%$) τότε $Επιτάχυνση < 10$, όσους επεξεργαστές N και αν χρησιμοποιήσουμε για την παράλληλη εκτέλεση.

Ο νόμος του Amdahl (3)

- Πράγματι, έστω ότι ποσοστό *s* του αλγορίθμου δεν παραλληλίζεται, τότε αν χρησιμοποιήσουμε *N* επεξεργαστές, ακόμη και αν υποθέσουμε ότι το υπόλοιπο *(1-s)* παραλληλίζεται τέλεια τότε:
- Χρόνος παράλληλης εκτέλεσης =

$$T_{παρ} = sT+(1-s)T/N$$

- οπότε

$$Επιτάχυνση = T/T_{παρ}$$
$$= T/(sT+(1-s)T/N)$$
$$= 1/(s+(1-s)/N)$$

Ο νόμος του Amdahl (4)

- Βλέπουμε ότι

$$Επιτάχυνση < 1/s$$

- και

$$Επιτάχυνση \rightarrow 1/s$$

όταν $N \rightarrow \infty$

- Συνεπώς το άνω όριο της επιτάχυνσης είναι το αντίστροφο του σειριακού τμήματος του προγράμματος, όσους επεξεργαστές κι αν διαθέτουμε.
- Πχ. Αν το 20% του προγράμματος είναι σειριακό, τότε $s=0.2$ και Επιτάχυνση < 1/0.2 = 5.

Ο νόμος του Amdahl - Παράδειγμα 1

- Επιθυμούμε να βελτιώσουμε την επίδοση ενός Web server και αποφασίσαμε να αντικαταστήσουμε την CPU με 10 παράλληλους επεξεργαστές. Οι μετρήσεις μας έδειξαν ότι το 70% ενός μέσου προγράμματος παραλληλίζεται τέλεια ενώ το υπόλοιπο 30% είναι I/O και δεν παραλληλίζεται. Τι επιτάχυνση πετυχαίνουμε?
- Απάντηση: Έχουμε $N=10$ επεξεργαστές. Το σειριακό κομμάτι που δεν παραλληλίζεται είναι $s=0.3$ (=30%). Συνεπώς:

$$Επιτάχυνση = 1/(s+(1-s)/N) = 1/(0.3+0.7/10) = 2.703$$

- Βλέπουμε ότι ενώ χρησιμοποιήσαμε 10 επεξεργαστές η επιτάχυνση που πετυχαίνουμε είναι μόλις 2.7 (σαφώς μικρότερη του 10...).

Ο νόμος του Amdahl - Παράδειγμα 2

- Επιθυμούμε να τριπλασιάσουμε την επίδοση ενός Web server και αποφασίσαμε να αντικαταστήσουμε την CPU με N παράλληλους επεξεργαστές. Οι μετρήσεις μας έδειξαν ότι το 80% ενός μέσου προγράμματος παραλληλίζεται τέλεια ενώ το υπόλοιπο 20% είναι I/O και δεν παραλληλίζεται. Πόσους επεξεργαστές χρειαζόμαστε?
- Απάντηση: Το $s = 0.2$ και θέλουμε $Επιτάχυνση = 3$

$$Επιτάχυνση = 1/(s + (1-s)/N)$$

$$3 = 1 / (0.2 + 0.8/N)$$

$$(0.2 + 0.8/N) = 1 / 3$$

$$0.8/N = 0.333 - 0.2$$

$$N = 0.8 / 0.1333 = 6$$