Επιστημονικός Υπολογισμός

Ε.Γαλλόπουλος

ΤΜΗΥΠ, Π. Πατρών

Διάλεξη 2: 11 Οκτωβρίου 2017

Περιεχόμενα

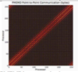
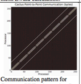
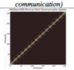
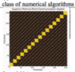
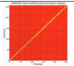
Berkeley Computational Dwarfs (Colella, et al.' 2004+)

A Dwarf is an algorithmic method that captures a pattern or motif of computations and communication which are believed to be the kernels for many future applications (A+06, Section 3)

- Dwarfs, which constitute classes where membership in a class is defined by similarity in computation and data movement. The dwarfs are specified at a high level of abstraction to allow reasoning about their behavior across a broad range of applications. Programs that are members of a particular class can be implemented differently and the underlying numerical methods may change over time, but the claim is that the underlying patterns have persisted through generations of changes and will remain important into the future.
- Dwarfs are designed to guide the development of parallel architectures and novel programming models.
- Ένα νέο σύστημα (υλικό και λογισμικό) θα αξιολογείται με βάση τις επιδόσεις της σε όλα τα dwarfs ώστε να θεωρείται ικανοποιητικά general purpose.
- Υπενθυμίζουμε ότι ιστορικά, πολλές αρχιτεκτονικές είχαν προταθεί ως special purpose.
- Η περίπτωση των GPUs, που ξεκίνησαν ως special purpose processors για γραφικά και κατέληξαν να χρησιμοποιούνται για general purpose HPC είναι ιδιαίτερα ενδιαφέρουσα.

Από τους 7 dwarfs[1] (motifs) του P. Colella στους 13 dwarfs του Berkeley

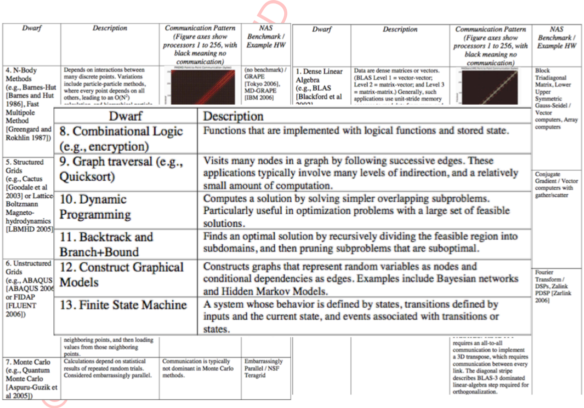
1. Structured grids
2. Unstructured grids
3. Dense linear algebra
4. Sparse linear algebra
5. Fast Fourier transforms
6. Particles
7. Monte Carlo

Dwarf	*Description*	*Communication Pattern (Figure axes show processors 1 to 256, with black meaning no communication)*	*NAS Benchmark / Example HW*
4. N-Body Methods (e.g., Barnes-Hut [Barnes and Hut 1986], Fast Multipole Method [Greengard and Rokhlin 1987])	Depends on interactions between many discrete points. Variations include particle-particle methods, where every point depends on all others, leading to an $O(N^2)$ calculation, and hierarchical particle methods, which combine forces or potentials from multiple points to reduce the computational complexity to $O(N \log N)$ or $O(N)$.	 PMEMD's communication pattern is that of a particle mesh Ewald calculation.	(no benchmark) / GRAPE [Tokyo 2006], MD-GRAPE [IBM 2006]
5. Structured Grids (e.g., Cactus [Goodale et al 2003] or Lattice-Boltzmann Magneto-hydrodynamics [LBMHD 2005])	Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality. Updates may be in place or between 2 versions of the grid. The grid may be subdivided into finer grids in areas of interest ("Adaptive Mesh Refinement"); and the transition between granularities may happen dynamically.	 Communication pattern for Cactus, a PDE solver using 7-point stencil on 3D block-structured grids.	Multi-Grid, Scalar Penta-diagonal / QCDOC [Edinburg 2006], BlueGeneL
6. Unstructured Grids (e.g., ABAQUS [ABAQUS 2006] or FIDAP [FLUENT 2006])	An irregular grid where data locations are selected, usually by underlying characteristics of the application. Data point location and connectivity of neighboring points must be explicit. The points on the grid are conceptually updated together. Updates typically involve multiple levels of memory reference indirection, as an update to any point requires first determining a list of neighboring points, and then loading values from those neighboring points.		Unstructured Adaptive / Vector computers with gather/scatter, Tera Multi Threaded Architecture [Berry et al 2006]
7. Monte Carlo (e.g., Quantum Monte Carlo [Aspuru-Guzik et al 2005])	Calculations depend on statistical results of repeated random trials. Considered embarrassingly parallel.	Communication is typically not dominant in Monte Carlo methods.	Embarrassingly Parallel / NSF Teragrid
1. Dense Linear Algebra (e.g., BLAS [Blackford et al 2002], ScaLAPACK [Blackford et al 1996], or MATLAB [MathWorks 2006])	Data are dense matrices or vectors. (BLAS Level 1 = vector-vector; Level 2 = matrix-vector; and Level 3 = matrix-matrix.) Generally, such applications use unit-stride memory accesses to read data from rows, and strided accesses to read data from columns.	 The communication pattern of MadBench, which makes heavy use of ScaLAPACK for parallel dense linear algebra, is typical of a much broader class of numerical algorithms	Block Triadiagonal Matrix, Lower Upper Symmetric Gauss-Seidel / Vector computers, Array computers
2. Sparse Linear Algebra (e.g., SpMV, OSKI [OSKI 2006], or SuperLU [Demmel et al 1999])	Data sets include many zero values. Data is usually stored in compressed matrices to reduce the storage and bandwidth requirements to access all of the nonzero values. One example is block compressed sparse row (BCSR). Because of the compressed formats, data is generally accessed with indexed loads and stores.	 SuperLU (communication pattern pictured above) uses the BCSR method for implementing sparse LU factorization.	Conjugate Gradient / Vector computers with gather/scatter
3. Spectral Methods (e.g., FFT [Cooley and Tukey 1965])	Data are in the frequency domain, as opposed to time or spatial domains. Typically, spectral methods use multiple butterfly stages, which combine multiply-add operations and a specific pattern of data permutation, with all-to-all communication for some stages and strictly local for others.	 PARATEC: The 3D FFT requires an all-to-all communication to implement a 3D transpose, which requires communication between every link. The diagonal stripe describes BLAS-3 dominated linear-algebra step required for orthogonalization.	Fourier Transform / DSPs, Zalink PDSP [Zarlink 2006]

[1] νάνους - οι πληροφορίες από (A+06, Section 3)

Από τους 7 dwarfs[1] (motifs) του P. Colella στους 13 dwarfs του Berkeley

1. Structured grids
2. Unstructured grids
3. Dense linear algebra
4. Sparse linear algebra
5. Fast Fourier transforms
6. Particles
7. Monte Carlo
8. Combinational Logic
9. Graph Traversal
10. Dynamic Programming
11. Backtracking
12. Probabilistic Graphical Models
13. Finite State Machines



Dwarf	Description
8. Combinational Logic (e.g., encryption)	Functions that are implemented with logical functions and stored state.
9. Graph traversal (e.g., Quicksort)	Visits many nodes in a graph by following successive edges. These applications typically involve many levels of indirection, and a relatively small amount of computation.
10. Dynamic Programming	Computes a solution by solving simpler overlapping subproblems. Particularly useful in optimization problems with a large set of feasible solutions.
11. Backtrack and Branch+Bound	Finds an optimal solution by recursively dividing the feasible region into subdomains, and then pruning subproblems that are suboptimal.
12. Construct Graphical Models	Constructs graphs that represent random variables as nodes and conditional dependencies as edges. Examples include Bayesian networks and Hidden Markov Models.
13. Finite State Machine	A system whose behavior is defined by states, transitions defined by inputs and the current state, and events associated with transitions or states.

[1] νάνους - οι πληροφορίες από (A+06, Section 3)

Modus operandi στα νέα συστήματα

Καλύτερα Υπολογισμοί παρά Επικοινωνία και Μεταφορές

Η επικοινωνία και μεταφορές δεδομένων είναι πολύ πιο αργές από τους υπολογισμούς.

Χρήση παραλληλίας

Μόνο με υλοποίηση και αξιοποίηση παραλληλίας μπορούμε να διατηρήσουμε το ρυθμό αύξησης της υπολογιστικής ισχύος.

Προσοχή στα ενεργειακά

Οι υπολογισμοί και οι μεταφορές (περισσότερο) είναι ενεργοβόροι και περιορίζουν τις αυξήσεις ισχύος.

Τα παραπάνω χαρακτηριστικά πρέπει να λαμβάνονται υπόψη στην ανάπτυξη εργαλείων του ΕΥ.

Υπολογιστικό μοντέλο

Στον πυρήνα της ιεραρχίας! Ένα υπολογιστικό μοντέλο περιγράφει μιαν *ιδεατή μηχανή* για την οποία μπορούμε να γράψουμε λογισμικό.

Παραδείγματα Turing machine, Random Access Machine (RAM), Random Access Stored Program (RASP) machine

Τι ζητάμε Μοντέλο που βοηθά στην πρόβλεψη της επίδοσης και παρέχει πληροφορίες για την βελτίωσή της.

Πρόκληση Δεν είναι εύκολο!

- Η μηχανή κρύβει αρχιτεκτονικές λεπτομέρειες και αλλαγές που οφείλονται σε καθαρά τεχνολογικές εξελίξεις (π.χ. μικρότερο κύκλο)
- Η ιδεατή μηχανή πρέπει να είναι αρκετά συγκεκριμένη ώστε να επιτρέπει την εξαγωγή συμπερασμάτων σχετικών με την επίδοση των προγραμμάτων που γράφονται γι΄ αυτήν.
- Abstraction should not be an obstruction (Joseph Gorse, 2013)

Παρατηρήσεις

- Κλασικά μοντέλα (όπως το RAM) επικεντρώνονταν στο πλήθος των αριθμητικών πράξεων.
- Για πολλά χρόνια η προσπάθεια αφορούσε στην κατασκευή αλγορίθμων που ελαχιστοποιούσαν την αριθμητική πολυπλοκότητα και φραγμάτων.

Παρατηρήσεις

Στις σύγχρονες αρχιτεκτονικές: It's the memory stupid! - R. Sites, 1996

- Το μοντέλο RAM δεν αρκεί για να προβλέψει την επίδοση
- Το χάσμα μεταξύ της διάρκειας ενός κύκλου της ΚΜΕ και της ταχύτητας επικοινωνίας επεξεργαστή με τη μνήμη αυξάνει
- Παράδειγμα: Τυπικά, ο χρόνος που απαιτείται για να μεταφέρουμε δεδομένα από την κύρια μνήμη είναι 100 φορές περισσότερο από το χρόνο ενός κύκλου σε έναν πυρήνα επεξεργαστή.
- Πρόβλημα: Πώς μπορεί να μειωθεί η επιβάρυνση του κόστους;
- Πρόταση: Αναπτύσσοντας τεχνικές απόκρυψης (κόστους) μεταφορών.
- Πώς: Αποκαλύπτοντας και αξιοποιώντας την τοπικότητα των προγραμμάτων με την υποστήριξη υλικού, ιδίως της κρυφής μνήμης

Ζητήματα

Πόσο γρήγορα λύνεται το πρόβλημα? Συνήθως εννοούμε ποιός είναι ο χρόνος επίλυσης.
Ποιά είναι η ταχύτητα εκτέλεσης του προγράμματος από το υπολογιστικό σύστημα? Συνήθως εννοούμε τον ρυθμό εκτέλεσης, δηλ. το πλήθος των πράξεων που εκτελούνται ανά μονάδα χρόνου. Η μέτρηση αυτή συνήθως αφορά στο μέσο ρυθμό κατά την εκτέλεση, δηλαδή στο λόγο (πράξεις/χρόνος). Ενίοτε ενδιαφέρει η μεταβολή του ρυθμού καθώς εκτελούνται διαφορετικά τμήματα του κώδικα.

Μονάδα μέτρησης ταχύτητας

$\mathrm{Mflop/s}$ = Million floating-point operations per second (προφ. μέγκαφλοπς)

- Λόγω των εξελίξεων στους επεξεργαστές, συνηθίζεται πλέον να αναφερόμαστε σε $\mathrm{Gigaflop/s}$ (δισεκατομμύρια πράξεις α.κ.υ. το δευτερόλεπτο) ... ενώ βαδίζουμε (στο απώτερο μέλλον) προς $\mathrm{Petaflop/s}$, μεσοπρόθεσμα προς $\mathrm{(h)Exaflops}$
- Η μετρική αυτή θεωρεί ότι το πλήθος πράξεων στον ορισμό του ρυθμού εκτέλεσης είναι οι πράξεις α.κ.υ. δηλ. στο Ω.
- Όλοι οι επεξεργαστές σήμερα εκτελούν τις αριθμητικές πράξεις με πολύ υψηλούς ρυθμούς.
- Αυτό δεν σημαίνει ότι εκτελούν κάθε πρόγραμμα γρήγορα καθώς πρέπει να εκτελεστούν και άλλες πράξεις που δεν συνυπολογίζονται στο Ω π.χ. μεταφορές. Αυτές επηρεάζουν και μειώνουν το μέσο ρυθμό εκτέλεσης.

Παρατηρήσεις για τα μοντέλα RAM, RASP

... we combine LOAD and STORE into the arithmetic operations by replacing sequences such as LOAD a; ADD b; STORE c by c <- a+b (Aho et al.74)

Παράδειγμα

Listing 1: MATLAB Horner, $\Omega = 2n$

```matlab
s = a(n+1);
for j=n:-1:1
    s = s*x + a(j);
end;
```

- Δεν λαμβάνεται υπόψη το κόστος επικοινωνίας
- $\Rightarrow$ συχνά γίνονται λάθος προβλέψεις
- π.χ. αλγόριθμοι για το ίδιο πρόβλημα με το ίδιο πλήθος αριθμ. πράξεων που έχουν πολύ διαφορετική επίδοση!

4.1.4 MOD1d

Program MOD1D tries to generate memory bank conflicts in order to observe the effect on the performance of the machine. To this end the following kernel is performed:

```
      DO 10 I = 1, INCR*1000, INCR
         S = S + A(I)*B(I)
   10 CONTINUE
```

where INCR ranges from 1 to 32. The results are displayed in Figure 15.

INCR	Cray-2S Mflop/s	Cray Y-MP Mflop/s
1	114.9	236.0
2	51.3	228.3
3	114.6	234.1
4	32.4	142.0
5	114.9	229.9
6	51.5	229.6
7	114.9	233.4
8	32.6	117.2
9	114.9	233.2
10	51.6	228.9
11	114.0	233.8
12	32.0	143.2
13	114.9	234.4
14	51.6	228.4
15	114.9	234.6
16	32.6	62.9
17	114.9	236.1
18	51.3	229.1
19	114.6	220.2
20	32.4	143.1
21	114.9	231.9
22	51.5	230.1
23	114.9	234.6
24	32.8	116.3
25	114.9	234.5
26	51.6	228.8
27	114.9	235.6
28	32.6	142.2
29	114.9	233.8
30	51.6	229.1
31	114.9	235.0
32	32.6	32.4

Figure 15. Performance of the inner product kernel for different values of the increment.

Απλό υπολογιστικό μοντέλο

Χαρακτηριστικά

- Επεξεργαστής και αρχιτεκτονική Load/Store
- αρχείο καταχωρητών
- κρυφή μνήμη ενός επιπέδου K θέσεων με write back
- κύρια μνήμη M θέσεων
- κόστη: Load, Store, πράξεις α.κ.υ.
- Κάθε πράξη αριθμ.κ.υ. στοιχίζει $\tau_{\text{αρθ}}$
- load από μνήμη στον επεξεργαστή σε χρόνο $\tau_{\text{μετ}}$
- load από κρυφή μνήμη στον επεξεργαστή σε $\tau_{\text{μετ}}^{(0)}$
- store από κρυφή μνήμη η επεξεργαστή σε $\tau_{\text{μετ}}$
- $\tau_{\text{μετ}}^{(0)} \approx 0$

Δείκτες

What we can't measure we can't improve (D. Patterson)

Ω αριθμός πράξεων α.κ.υ.

Φ αριθμός μεταφορών μεταξύ κύριας μνήμης και καταχωρητών ή κρυφής μνήμης

Φ_{min} ελάχιστος αριθμός μεταφορών αλγορίθμου αν διαθέταμε απεριόριστη μνήμη σε όλα τα επίπεδα

$T_{\alpha\rho\theta}$ χρόνος που αναλώνεται για αριθμητικές πράξεις α.κ.υ.

$T_{\mu\epsilon\tau}$ χρόνος που αναλώνεται για μεταφορές α.κ.υ.

Υποθέτουμε ότι ο χρόνος εκτέλεσης μιας υλοποίησης μπορεί να εκτιμηθεί από τον τύπο

$$\boxed{\;= T_{\alpha\rho\theta} + T_{\mu\epsilon\tau}}$$

και ότι έχουμε μόνον 2 επίπεδα μνήμης: Κύρια μνήμη και register - cache file.

Παρατηρήσεις

Θέτουμε

$\mu := \frac{\Phi}{\Omega}$	μεταφορές ανά αριθμητική πράξη για τη συγκεκριμένη υλοποίηση (θέμα λογισμικού)
$\tau_{\text{αρθ}}$	χρόνος για 1 αριθμ. πράξη (θέμα υλικού)
$\tau_{\text{μετ}}$	χρόνος για 1 μεταφορά (θέμα υλικού)

Εμπειρική παρατήρηση και υπόθεση εργασίας:

Οι μεταφορές είναι πολύ πιο ακριβές από τις αριθμητικές πράξεις

$$\tau_{\text{μετ}} \gg \tau_{\text{αρθ}}$$

Επακόλουθα

Ξαναγράφουμε

$$
\begin{aligned}
T &= T_{\text{αρθ}} + T_{\text{μετ}} \\
&= \tau_{\text{αρθ}}\Omega + \tau_{\text{μετ}}\Phi, \\
&= T_{\text{αρθ}}\left(1 + \mu\frac{\tau_{\text{μετ}}}{\tau_{\text{αρθ}}}\right),
\end{aligned}
$$

Δείκτες

Αν διερευνήσουμε όλες τις υλοποιήσεις του αλγορίθμου, θα υπάρχει κάποια που απαιτεί το μικρότερο κόστος: Γράφουμε για το αντίστοιχο μ,

$$\mu_{\text{best}} = \arg \min_{[\Omega,\Phi]\in \text{ σύνολο υλοποιήσεων}} T(\mu)$$

Προσοχή: Δεν έχουμε ακόμα μιλήσει για την ακρίβεια!

K. Asanovic et al.
The landscape of parallel computing research: A view from Berkeley.
Technical report no. ucb/eecs-2006-183, University of California at Berkeley, Dec. 2006.
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.