PRINCIPLES OF SCALABLE PERFORMANCE

3.1 PERFORMANCE METRICSAND MEASURES

Parallelism Profile in Programs

Degree of Parallelism reflects the extent to which software parallelism matches hardware parallelism. the execution of a program on a parallel computer may use different numbers of processors at different time periods during the execution cycle. For each time period, the number of processors used to execute a program is defined as the degree of parallelism (DOP).

The plot of the DOP as a function of time is called the parallelism profile of a given program.The DOP was definedunder the assumption of that, unbounded number of available processors and other necessary resources.The DOP may not always be achievable on a real computer with limited resources.

Average parallelism

We consider a parallel computer consisting of n homogeneous processors. The maximum parallelism in a profile is m. In the ideal case, n>> m. The computing capacity Δ of a single processor is approximated by the execution rate, sueh as MIPS without consideringoverhead. When i processors are busyduring an observation period, we have DDP = i.

The total amount of work W (instructions or computations) performed is proportional to th the profile curve:

$$W = \Delta \int_{t_1}^{t_2} DOP(t)\, dt$$

This integral is often computed with the following discrete summation:

$$W = \Delta \sum_{i=1}^{m} i \cdot t_i$$

where t_i is the total amount of time that DOP = i and $\sum_{i=1}^{m} t_i = t_2 - t_1$ is the total elapsed time.

The *average parallelism* A is computed by

$$A = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} DOP(t)\, dt$$

In discrete form, we have

$$A = \left(\sum_{i=1}^{m} i \cdot t_i \right) \Big/ \left(\sum_{i=1}^{m} t_i \right)$$

Example 3.1 Parallelism profile and average parallelism of a divide-and-conquer algorithm (Sun and Hi,1993)

As illustrated in Fig. 3.1, the parallelism profile of a divide-and-conquer algorithm increases from l to itspeak value m = 8 and then decreases to 0 during the observation period (t1, t2).

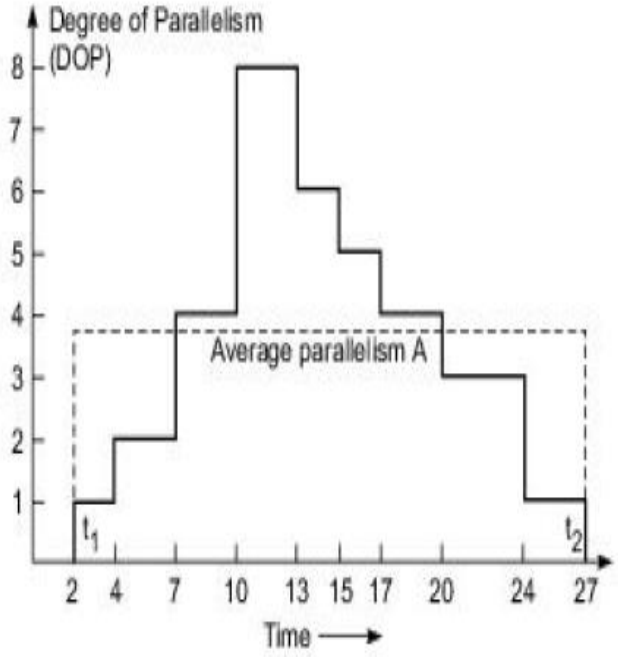

Fig. 3.1 Parallelism profile of a divide-and-conquer algorithm

In Fig. 3.1, the average parallelism $A = (1 \times 5 + 2 \times 3 + 3 \times 4 + 4 \times 6 + 5 \times 2 + 6 \times 2 + 8 \times 3)/(5 + 3 + 4 + 6 + 2 + 2 + 3) = 93/25 = 3.72$. In fact, the total workload $W = A\Delta(t_2 - t_1)$, and A is an upper bound of the asymptotic speedup to be defined below.

Asymptotic Speedup Denote the amount of work executed with DOP = i as $W_i = i\Delta t_i$ or we can write $W = \sum_{i=1}^{m} W_i$. The execution time of W_i on a single processor (sequentially) is $t_i(1) = W_i/\Delta$. The execution time of W_i on k processors is $t_i(k) = W_i/k\Delta$. With an infinite number of available processors, $t_i(\infty) = W_i/i\Delta$ for $1 \le i \le m$. Thus we can write the *response time* as

$$T(1) = \sum_{i=1}^{m} t_i(1) = \sum_{i=1}^{m} \frac{W_i}{\Delta} \qquad (3.5)$$

$$T(\infty) = \sum_{i=1}^{m} t_i \ (\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta} \tag{3.6}$$

The *asymptotic speedup* S_∞ is defined as the ratio of $T(1)$ to $T(\infty)$:

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i / i} \tag{3.7}$$

Comparing Eqs. 3.4 and 3.7, we realize that $S_\infty = A$ in the ideal case. In general, $S_\infty \leq A$ if communication latency and other system overhead are considered. Note that both S_∞ and A are defined under the assumption $n = \infty$ or $n >> m$.

Mean Performance

Consider a parallel computer with n processors executing m programs in various modes with different performance levels. We want to define the mean performance of such multimode computers. With a weight distribution we can define a meaningful performance expression.

Different execution modes may correspond to scalar, vector, sequential, or parallel processing. Each program may be executed with a combination of these modes. Harmonic-mean performance provides an average performance across a large number of programs running in various modes.

Arithmetic Mean Performance Let $\{R_i\}$ be the execution rates of programs $i = 1, 2, \ldots, m$. The *arithmetic mean execution rate* is defined as

$$R_a = \sum_{i=1}^{m} R_i / m \tag{3.8}$$

The expression R_a assumes equal weighting $(1/m)$ on all m programs. If the programs are weighted with a distribution $\pi = \{f_i | i = 1, 2, \ldots, m\}$, we define a *weighted arithmetic mean execution rate* as follows:

$$R_a^* = \sum_{i=1}^{m} (f_i R_i) \tag{3.9}$$

Arithmetic mean execution rate is proportional to the sum of the inverses of execution times; it is not inversely proportional to the sum of execution times. Consequently, the arithmetic mean execution rate fails to represent the real times consumed by the benchmarks when they are actually executed.

Harmonic Mean Performance With the weakness of arithmetic mean performance measure, we need to develop a mean performance expression based on arithmetic mean execution time. In fact, $T_i = 1/R_i$ is the mean execution time per instruction for program i. The *arithmetic mean execution time* per instruction is defined by

$$T_a = \frac{1}{m} \sum_{i=1}^{m} T_i = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{R_i} \tag{3.10}$$

The *harmonic mean execution rate* across m benchmark programs is thus defined by the fact $R_h = 1/T_a$:

$$R_h = \frac{m}{\sum_{i=1}^{m} (1/R_i)} \tag{3.11}$$

Therefore, the harmonic mean performance is indeed related to the average execution time. With a weight distribution $\pi = \{f_i | i = 1, 2, \ldots, m\}$, we can define the *weighted harmonic mean execution rate as*:

$$R_h^* = \frac{1}{\sum_{i=1}^{m} (f_i / R_i)} \tag{3.12}$$

The above harmonic mean performance expressions correspond to the total number of operations divided by the total time. Compared to arithmetic mean, the harmonic mean execution rate is closer to the real performance.

Harmonic Mean Speedup Another way to apply the harmonic mean concept is to tie the various modes of a program to the number of processors used. Suppose a program (or a workload of multiple programs combined) is to be executed on an n-processor system. During the executing period, the program may use $i = 1, 2, \ldots, n$ processors in different time periods.

We say the program is executed in *mode i*, if i processors are used. The corresponding execution rate R_i is used to reflect the collective speed of i processors. Assume that $T_1 = 1/R_1 = 1$ is the sequential execution time on a uniprocessor with an execution rate $R_1 = 1$. Then $T_i = 1/R_i = 1/i$ is the execution time of using i processors with a combined execution rate of $R_i = i$ in the ideal case.

Suppose the given program is executed in n execution modes with a weight distribution $w = \{f_i | i = 1, 2, \ldots, n\}$. A *weighted harmonic mean speedup* is defined as follows:

$$S = T_1/T^* = \frac{1}{(\sum_{i=1}^{n} f_i/R_i)} \tag{3.13}$$

where $T^* = 1/R_h^*$ is the *weighted arithmetic mean execution time* across the n execution modes, similar to that derived in Eq. 3.12.

Example 3.2 Harmonic mean speedup for a multiprocessor operating in n execution modes

(Hwarlg and Briggs, 19:14)

In Fig. 3.2, we plot Eq. 3.13 based on the assumption that $T_i = 1/i$ for all $i = 1, 2, \ldots, n$. This corresponds to the ideal case in which a unit-time job is done by i processors in minimum time. The assumption can also be interpreted as $R_i = i$ because the execution rate increases i times from $R_1 = 1$ when i processors are fully utilized without waste.

The three probability distributions π_1, π_2, and π_3 correspond to three processor utilization patterns. Let $s = \sum_{i=1}^{n} i$. $\pi_1 = (1/n, 1/n, \ldots, 1/n)$ corresponds to a uniform distribution over the n execution modes, $\pi_2 = (1/s, 2/s, \ldots, n/s)$ favors using more processors, and $\pi_3 = (n/s, (n-1)/s, \ldots, 2/s, 1/s)$ favors using fewer processors.

The ideal case corresponds to the 45° dashed line. Obviously, π_2 produces a higher speedup than π_1 does. The distribution π_1 is superior to the distribution π_3 in Fig. 3.2.

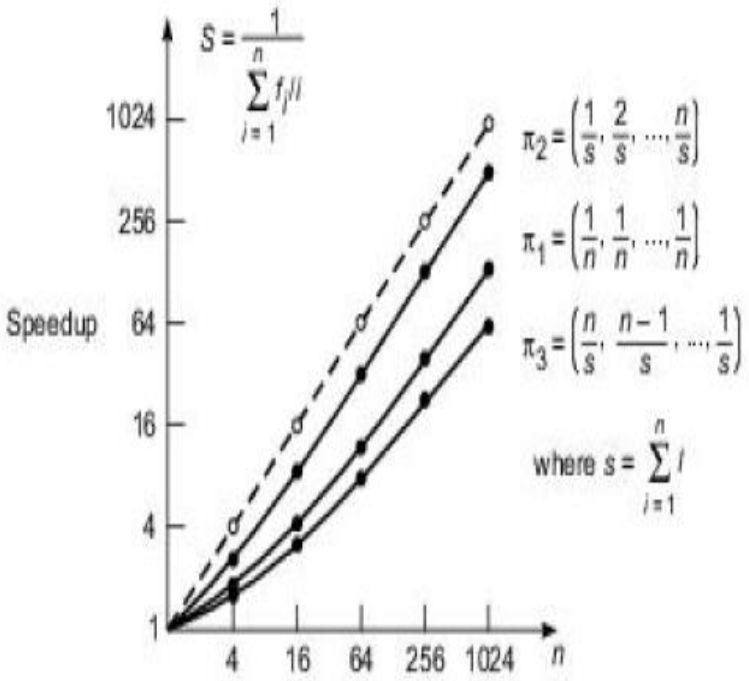

Fig. 3.2 Harmonic mean speedup performance with respect to three probability distributions: π_1 for uniform distribution, π_2 in favor of using more processors, and π_3 in favor of using fewer processors

Amdahl's Law Using Eq. 3.13, one can derive Amdahl's law as follows: First, assume $R_i = i$, $w = (\alpha, 0, 0, \ldots, 0, 1-\alpha)$; i.e., $w_1 = \alpha$, $w_n = 1-\alpha$, and $w_i = 0$ for $i \neq 1$ and $i \neq n$. This implies that the system is used either in a pure sequential mode on one processor with a probability α, or in a fully parallel mode using n processors with a probability $1-\alpha$. Substituting $R_1 = 1$ and $R_n = n$ and w into Eq. 3.13, we obtain the following speedup expression:

$$S_n = \frac{n}{1+(n-1)\alpha} \tag{3.14}$$

This is known as Amdahl's law. The implication is that $S \to 1/\alpha$ as $n \to \infty$. In other words, under the above probability assumption, the best speedup one can expect is upper-bounded by $1/\alpha$, regardless of how many processors are employed.

In Fig. 3.3, we plot Eq. 3.14 as a function of n for four values of α. When $\alpha = 0$, the ideal speedup is achieved. As the value of α increases from 0.01 to 0.1 to 0.9, the speedup performance drops sharply.

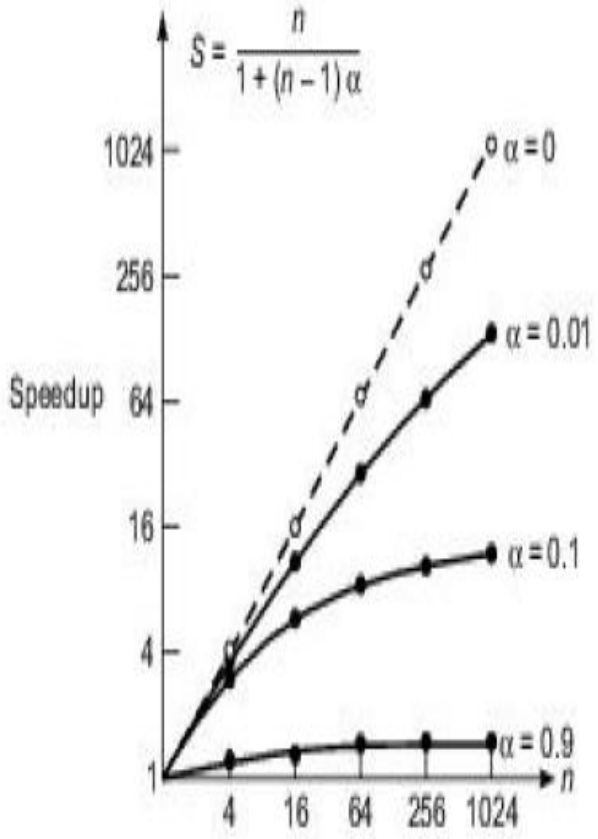

Fig. 3.3 Speedup performance with respect to the probability distribution $\pi = (\alpha, 0, \ldots, 0, 1-\alpha)$ where α is the fraction of sequential bottleneck

3.1.3 Efficiency, Utilization, and Quality

Ruby Lee (1980) has defined several parameters for evaluating parallel computations. These are fundamental concepts in parallel processing. Tradeoffs among these performance factors are often encountered in real-life applications.

System Efficiency Let $O(n)$ be the total number of unit operations performed by an n-processor system and $T(n)$ be the execution time in unit time steps. In general, $T(n) < O(n)$ if more than one operation is performed by n processors per unit time, where $n \geq 2$. Assume $T(1) = O(1)$ in a uniprocessor system. The *speedup factor* is defined as

$$S(n) = T(1)/T(n) \tag{3.15}$$

The *system efficiency* for an n-processor system is defined by

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{nT(n)} \tag{3.16}$$

Efficiency is an indication of the actual degree of speedup performance achieved as compared with the maximum value. Since $1 \leq S(n) \leq n$, we have $1/n \leq E(n) \leq 1$.

The lowest efficiency corresponds to the case of the entire program code being executed sequentially on a single processor, the other processors remaining idle. The maximum efficiency is achieved when all n processors are fully utilized throughout the execution period.

Redundancy and Utilization The *redundancy* in a parallel computation is defined as the ratio of $O(n)$ to $O(1)$:

$$R(n) = O(n)/O(1) \tag{3.17}$$

This ratio signifies the extent of matching between software parallelism and hardware parallelism. Obviously $1 \leq R(n) \leq n$. The *system utilization* in a parallel computation is defined as

$$U(n) = R(n)E(n) = \frac{O(n)}{nT(n)} \tag{3.18}$$

The system utilization indicates the percentage of resources (processors, memories, etc.) that was kept busy during the execution of a parallel program. It is interesting to note the following relationships: $1/n \leq E(n) \leq U(n) \leq 1$ and $1 \leq R(n) \leq 1/E(n) \leq n$.

Quality of Parallelism The *quality* of a parallel computation is directly proportional to the speedup and efficiency and inversely related to the redundancy. Thus, we have

$$Q(n) = \frac{S(n)E(n)}{R(n)} = \frac{T^3(1)}{nT^2(n)O(n)} \tag{3.19}$$

Since $E(n)$ is always a fraction and $R(n)$ is a number between 1 and n, the quality $Q(n)$ is always upper-bounded by the speedup $S(n)$.

Example 3.3 A hypothetical workload and performance plots

In Fig. 3.4, we compare the relative magnitudes of $S(n)$, $E(n)$, $R(n)$, $U(n)$, and $Q(n)$ as a function of machine size n, with respect to a hypothetical workload characterized by $O(1) = T(1) = n^3$, $O(n) = n^3 + n^2 \log_2 n$, and $T(n) = 4n^3/(n+3)$.

Substituting these measures into Eqs. 3.15 to 3.19, we obtain the following performance expressions:

$$
\begin{aligned}
S(n) &= (n+3)/4 \\
E(n) &= (n+3)/(4n) \\
R(n) &= (n+\log_2 n)/n \\
U(n) &= (n+3)(n+\log_2 n)/(4n^2) \\
Q(n) &= (n+3)^2/(16(n+\log_2 n))
\end{aligned}
$$

The relationships $1/n \le E(n) \le U(n) \le 1$ and $0 \le Q(n) \le S(n) \le n$ are observed where the linear speedup corresponds to the ideal case of 100% efficiency.

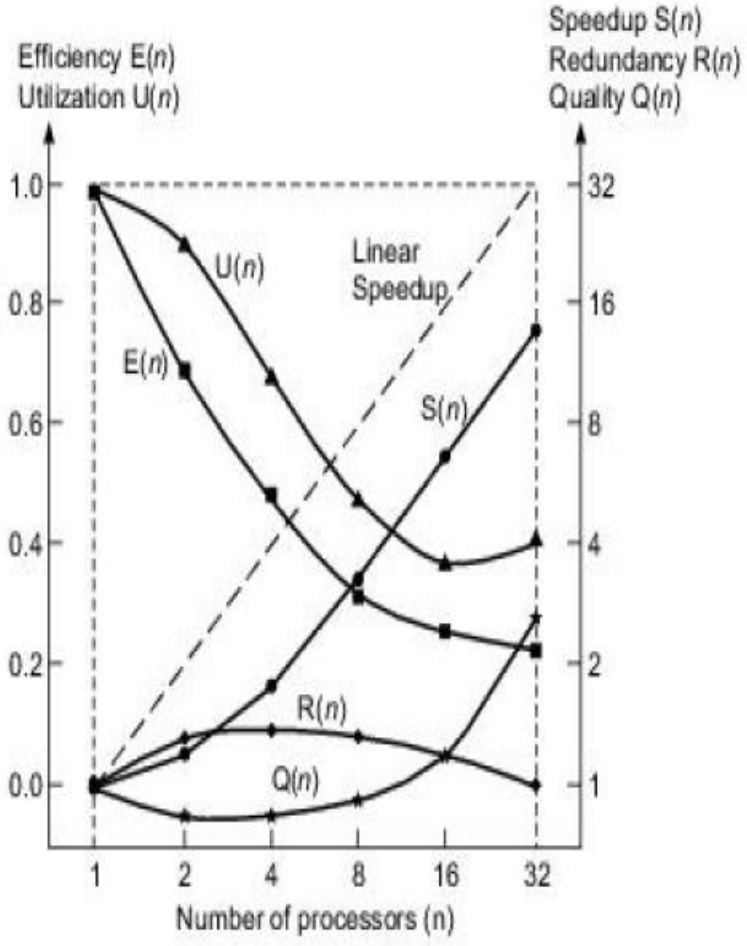

Fig. 3.4 Performance measures for Example 3.3 on a parallel computer with up to 32 processors

To- summarise the above discussion on performance indices, we use the speedup S(n) to indicate the degree of speed gain in a parallel computation. The efficiency E('n) measures the useful portion of the total work performed by n processors. The redundancy R(n) measures the extent of workload increase. The utilization U(n) indicates the extent to which resources are utilized during a parallel computation. Finally, the quality Q(n) combines the effects of speedup, efficiency, and redundancy into a single expression to assess the relative merit of a parallel computation on a computer system.

The speedup and efficiency of 10 parallel computers are reported in below Table 3.1 for solving a linear system of I000 equations.

Table 3.1 *Speedup and Efficiency of Parallel Computers for Solving a Linear System with 1000 Unknowns*

Computer Model	*No. of Processors* n	*Uni-processor Timing* T_1 (s)	*Multi-processor Timing* T_n (s)	*Speedup* $S = T_1/T_n$	*Efficiency* $E = S/n$
Cray Y-MP C90	16	0.77	0.069	11.12	0.69
NEC SX-3	2	0.15	0.082	1.82	0.91
Cray Y-MP/8	8	2.17	0.312	6.96	0.87
Fujitsu AP 1000	512	160.0	1.10	147.0	0.29
IBM 3090/600S VF	6	7.27	1.29	5.64	0.94
Intel Delta	512	22.0	1.50	14.7	0.03
Alliant FX/2800-200	14	22.9	2.06	11.1	0.79
nCUBE/2	1024	331.0	2.59	128.0	0.12
Convex C3240	4	14.9	3.92	3.81	0.95
Parsytec FT-400	400	1075.0	4.90	219.0	0.55

Benchmarks and Performance Measures

In practice, the MIPS and Mflops ratings and other performance indicators to be should be measured from Running benchmarks or real programs on real machines.

The following are some of the benchmarks used for measuring performance:

The Dhrynone Result

This is a CPU-intensive benchmark consisting of a mix of about 100 highlevellanguage instructions and data types found in system programming applications where floating-pointoperations are not used . The Dbrystone statements are balanced with respect to statementtype, data type, and locality of reference, with no operating system calls and making no use of libraryfunctions or subroutines. Thus the Dhrystone rating should be a measure of the integer performance ofmodem processors. The unit KDhrystones is often used in reporting Dhrystone results.

Whetstone Benchmark

This is a Fortran-based synthetic benchmark assessing the floating-pointperformance, measured in the number of KWhetstoness that a system can perform. The benchmark includesboth integer and floating-point operations involving array indexing, subroutine calls, parameter passing,conditional branching, and trigonometric functions. The does not contain any vectorizable code and shows dependence on the system'smathematics library and efiiciency ofthe code generated by a compiler.Both the Dhrystone and Whetstone are synthetic benchmarks whose perfomiance results depend heavilyon the compilers used.

The TPS and KLIPS Rating:

On-line transaction processing applications demand rapid, interactiveprocessing for a large number of relatively simple transactions. Automated teller machines and airline reservation systems are familiar examples. The throughput ofcomputers for on-line transaction processing is often measured in TPS transactions per second.For such web applications, application-specific benchmarks have become more important than general purposebenchmarks such as Whetstone. For web servers providing 24 x7 service require to benchmark which is under simulated but realistic load conditions and performance parameters such as: throughput innumber of requests served and average response time.

ln artificial intelligence applications, the measure KLIPS (kilo logic inference per second) was used atone time to indicate the reasoning power of an AI machine. For example, the high-speed inference machine developed under Japan's Fifth Generation Computer System Project claimed a performance of 400 KLIPS.Assuming that each logic inference operation involves about 100 assembly instructions, 400 KLIPSimplies approximately 40 MIPS in this sense.

3.2 PARALLEL PROCESSING APPLICATIONS

Massive Parallelism for Grand Challenges

Grand Challenges:

1) Thc magnetic recording industry relys on the use of computers to study magmetostatic and exchange interactions in order to reduce noise in metallic thin films used to coat high-density disk. ln genral all research in science and enginnering makes heavy demands on computing power.

(2) Rational drug design is being aided by computers in the search for a cure for cancer, acquired immunodeficiency syndrome and other diseases. Using a high-performance computer, new potential agents have been identified that block the action of human immunodeficiency virus protease.

(3) Design of high-speed transport aircraft is being aided by computational fluid dynamics running on supercomputers. Fuel combustion can be made more efficient by designing better engine models through chemical kinetics calculations.

(4) Catalysts for chemical reactions are being designed with computers for many biological processes which are catalytically controlled by enzymes. Massively parallel quantum models demand large simulations to reduce the time required to design catalysts and to optimize their properties.

(5) Ocean modeling cannot be accurate without supercomputing MPP systems. Ozone depletion and climate research demands the use of computers in analyzing the complex thermal, chemical and fluid-dynamic mechanisms involved.

(6) Other important areas demanding computational support include digital anatomy in real-time medical diagnosis, air pollution reduction through computational modeling, the design of protein structures by computational biologists, image processing and understanding, and technology linking research to education.