QUESTION 1

i. Application is 80% parallelizable. Therefore 40% of the instructions can be executed in parallel on the second core.

$$Performance_{old} = \frac{1}{Execution\,Time_{old}} = \frac{clock\ rate_{old}}{instructions * CPI}$$

$$Performance_{new} = \frac{clock\ rate_{new}}{(0.6 * instructions) * CPI}$$

For $Performance_{old} = Performance_{new}$ we get,

$$clock\ rate_{new} = 0.6 * clock\ rate_{old}$$

i.e. a 40% decreased clock rate

ii. Since voltage decreases linearly with frequency, the dual core system will run at 0.6 times the original voltage. Assuming that the capacitive load is constant for both configurations,

$$\begin{aligned} Power_{old} &= \frac{1}{2} * capacitive\,load * voltage^2 * clock\ rate \\ Power_{new} &= \frac{1}{2} * capacitive\,load * (0.6 * voltage)^2 * (0.6 * clock\ rate) \\ &= 0.6^3 * Power_{old} \\ &= 0.216 * Power_{old} \end{aligned}$$

Thus, the **dual core system requires 78.4% less dynamic power** as compared to the single core system.

iii. Since voltage floor is 75%, the floor for clock rate reduction is also 75%.

$$\frac{clock\ rate_{old}}{instructions_{old} * CPI} = \frac{0.75 * clock\ rate_{old}}{instructions_{new} * CPI}$$

$$instructions_{new} = 0.75 * instructions_{old}$$

Therefore,
required % parallelization = 2*0.25*100 = **50%**

QUESTION 2

i.

$$MIPS_{oldConfig} = \frac{I+(F*Y)}{W*10^6}$$

$$MIPS_{newConfig} = \frac{I+F}{B*10^6}$$

ii.

$$I = (MIPS_{oldConfig}*W*10^6)-(F*Y)$$
$$= (120*4*10^6)-(8*10^6*50)$$
$$\mathbf{= 80 * 10^6}$$

iii.

$$B = \frac{I+F}{MIPS_{newConfig}*10^6} = \frac{(80*10^6)+(8*10^6)}{80*10^6} \quad \mathbf{= 1.1\ sec}$$

iv. To calculate MFLOPS rating we need to find the time spent by the new configuration to execute only floating point instructions. We know that time spent executing integer instructions is equal for both the old and new configuration since the ALU is unchanged and only a FP co-processor is added.

$$Time_{intInstr} = \frac{W*I}{I+(F*Y)}$$
$$= \frac{4*80*10^6}{(80*10^6)+(8*10^6*50)}$$
$$= \frac{2}{3} sec$$

$$Time_{floatInstr} = B - Time_{intInstr}$$
$$= 1.1-\frac{2}{3}$$
$$= 0.4333 sec$$

$$MFLOPS_{newConfig} = \frac{F}{Time_{floatInstr}*10^6} = \frac{8*10^6}{0.4333*10^6} \quad \mathbf{= 18.4630}$$

v. Although $MIPS_{oldConfig} > MIPS_{newConfig}$ the execution of the benchmark is 72.5% faster on the new configuration than the old one. The MIPS rating of the old configuration is greater than that of the new one because of the emulation factor of 50 for floating point instructions. The performance of integer instructions on both configurations is constant. The decision should be based on floating point instruction performance of both configurations. **Based on execution times the new configuration is clearly the winner in this case.**

Also, let us calculate MFLOPS for the old configuration.

$$\begin{aligned} MFLOPS_{oldConfig} &= \frac{F}{(W - Time_{intInstr}) * 10^6} \\ &= \frac{8 * 10^6}{(4 - \frac{2}{3}) * 10^6} \\ &= 2.4 \end{aligned}$$

The MFLOPS rating for the new configuration too is substantially higher than that of the old configuration, which reflects the performance of both configurations for floating point instructions.

QUESTION 3

A)

$$ExecTime_{vectorMode} = ExecTime_{normalMode} * \{(1 - \frac{\%Vectorization}{100}) + \frac{\%Vectorization}{100 * 20}\}$$

$$Speedup = \frac{ExecTime_{normalMode}}{ExecTime_{vectorMode}} = \frac{1}{1 - [\frac{19}{20} * \frac{\%Vectorization}{100}]}$$

Therefore,

$$\%Vectorization = [1 - \frac{1}{Speedup}] * \frac{20 * 100}{19}$$

For a speedup of 2, we get,

$$\%Vectorization = [1 - \frac{1}{2}] * \frac{20 * 100}{19}$$

$$= \mathbf{52.632\%}$$

B) It is given that maximum speedup achievable is 20.
For a speedup of 10 (half of the maximum),

$$\%Vectorization = [1 - \frac{1}{10}] * \frac{20 * 100}{19}$$

$$= \mathbf{94.73\%}$$

C) The generalized equation for new execution time is,

$$ExecTime_{vectorMode} = ExecTime_{normalMode} * \{(1 - \frac{\%Vectorization}{100}) + \frac{\%Vectorization}{100 * Vector\ Rate}\}$$

If the vector rate is doubled to 40, holding the percent of vectorization constant, the improved execution time will be,

$$ExecTime_{vectorRate=40} = ExecTime_{normalMode} * \{(1 - \frac{70}{100}) + \frac{70}{100 * 40}\}$$

$$= ExecTime_{normalMode} * 0.3175$$

If we were to achieve the above improved execution time by increasing the percent of vectorization while holding the vector rate constant at 20, the improved percent vectorization can be found by solving,

$$ExecTime_{normalMode} * 0.3175 = ExecTime_{normalMode} * \{(1 - \frac{\%Vectorization}{100}) + \frac{\%Vectorization}{100 * 20}\}$$

$$\%Vectorization = (1 - 0.3175) * \frac{100 * 20}{19} = 71.8421\%$$

Thus, equal performance improvement can be achieved by 1.8421% increase in the percentage of vectorization. This implies **equal performance improvement is possible by increasing percentage vectorization through lower investment in terms of time and money as compared to vector rate enhancement.**

QUESTION 4

A) As a result of using the new addressing mode for 10% of the displacement load and store instructions,

$$\#\,instructions\ \ affected\ =\ 3.71\,\%$$

Since number of instructions for specifying the address computation and offset displacement is halved as a result of the new addressing mode,

$$\#\,instructions\ \ decreased\ =\ 1.855\,\%$$

Therefore the ratio,

$$\frac{\#\,instructions_{new}}{\#\,instructions_{old}} = \frac{0.98145}{1}$$

B) Since the introduction of the new addressing mode causes the clock cycle duration to increase by 5%, the new execution time is,

$$\begin{aligned} ExecTime_{new} &= ExecTime_{old} * 0.98145 * 1.05 \\ &= ExecTime_{old} * 1.0305 \end{aligned}$$

that is, a 3.05% increase in execution time.

The old machine is faster than the new one by 3.05%

QUESTION 5

Let us consider the following excerpt of the data on offset sizes for this question,

Offset Bits	Signed Offset Bits	Cum. Data Ref.	Cum. Branches
0	0	30.4%	0.1%
7	8	66.9%	85.2%
15	16	100%	98.5%
23	24	100%	100%

We can also group the data on instruction usage by programs in the benchmark as follows,

Instruction Class	Instructions	Avg. Usage
Data Transfer / Reference	Load, store, load imm	38%
Control / Branch	Cond branch, cond move, jump, call, return	16%
ALU	Rest of the instructions	46%

The "load imm" instruction is classified as a data transfer/reference instruction since the immediate value requires additional bits in the instruction which can be considered to be offset length.

The register-register move instruction usage has been stated to be considered in the MIPS table as part of the "or" instruction usage. That leaves the "cond move" instruction which we classify as a control/branch instruction. This instruction does not need offset bits as the move is between GPRs, and such a case is assumed to be considered in the 0 offset length table row/entry.

A)

$$\begin{aligned} AvgInstrLength_{variableOffset} &= (0.46*16) \\ &+ 0.38*[(0.304*16)+(0.365*24)+(0.331*32)] \\ &+ 0.16*[(0.001*16)+(0.851*24)+(0.133*32)+(0.015*40)] \\ &= \mathbf{20.6094\ bits} \end{aligned}$$

$$\boldsymbol{AvgInstrFetch_{variableOffset}} \sim \mathbf{3\ bytes}$$

B) Offset length greater than 8 bits causes a new instruction to be used to specify the rest of the offset. The number of instructions is determined by calculating $\lceil offset\,length/8 \rceil$

$$\begin{aligned} AvgInstrLength_{fixedInstr} &= (0.46*24) \\ &+ 0.38*[(0.669*24)+(0.331*48)] \\ &+ 0.16*[(0.852*24)+(0.133*48)+(0.015*72)] \end{aligned}$$

$= 11.04 + 12.13872 + 4.46592$

$\mathbf{= 27.64464\ bits}$

$\boldsymbol{AvgInstrFetch_{fixedInstr} \sim 4\ \text{bytes}}$

If the configuration fetches instructions byte-by-byte, variable offset length (as in part A) is better than fixed instruction length as it will cause one less byte to be fetched. On the other hand if the configuration fetches instructions word-by-word (word-length > 1 byte) then both variable offset length and fixed instruction length are rendered equivalent as same number of words will be fetched during instruction fetch.

C) If length of instructions is determined by class of instructions i.e. branch and data transfer instructions are assumed to use offset bits (disregarding that some may not specify offset),

$$AvgInstrLength_{fixedOffset} = (0.46*16) + [(0.38+0.16)*40]$$

$\mathbf{= 28.96\ bits}$

$\boldsymbol{AvgInstrFetch_{fixedOffset} \sim 4\ \text{bytes}}$

If the configuration fetches instructions byte-by-byte, variable offset length (as in part A) is better than fixed offset length as it will cause one less byte to be fetched. Irrespective of whether the configuration fetches instructions byte-by-byte or word-by-word (word-length > 1 byte), both fixed instruction length and fixed offset length are equivalent as same number of bytes/words will be fetched during instruction fetch.

On the other hand if the configuration optimizes length of instructions by not allocating 24 offset bits to branch and data transfer instructions that do not specify offset we get,

$$\begin{aligned} AvgInstrLength_{fixedOffset} &= (0.46*16) \\ &\quad + 0.38*[(0.304*16)+(0.696*40)] \\ &\quad + 0.16*[(0.001*16)+(0.999*40)] \end{aligned}$$

$\mathbf{= 26.18\ bits}$

$\boldsymbol{AvgInstrFetch_{fixedOffset} \sim 4\ \text{bytes}}$

If the configuration fetches instructions byte-by-byte, variable offset length (as in part A) is still better than fixed offset length as it will cause one less byte to be fetched.
In terms of bits fixed offset length results in shorter average instruction length than fixed instruction length in this case. Nevertheless, irrespective of whether the configuration fetches instructions byte-by-byte or word-by-word (word-length > 1 byte), both fixed instruction length and fixed offset length are equivalent as same number of bytes/words will be fetched during instruction fetch.